UiO : Department of Informatics
University of Oslo

Investigating data confidentiality in cloud computing

Kim Fredrik Olsen

Master's Thesis Spring 2014

Investigating data confidentiality in cloud computing

Kim Fredrik Olsen

20th May 2014

Abstract

Public cloud services have seen a massive growth in recent years as organizations have started to move their infrastructure to the cloud. This enables the organizations to take advantage of the dynamic infrastructure and reduced infrastructure costs that cloud computing brings. Public cloud services also bring new security challenges however, as the infrastructure is running as instances in a shared environment.

The goal of this thesis is to explore file confidentiality challenges in the cloud. This is done in practice by developing a prototype that can extract file(s) from ongoing file transfers occurring on a virtual machine by analyzing its memory from the physical machine (VMM).

The prototype shows that it is possible to extract the files, but that the success rate depend on a number of factors such as the network speed the file(s) are downloaded at and the size of the files.

Throughout the thesis these factors have been measured and analyzed in order to better understand how they affect the prototype and the extraction of files from file transfers in general.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Ismail Hassan, for his guidance, encouragement and genuine interest in the thesis throughout the whole period.

Secondly i would like to offer my gratitude to Oslo and Akershus University College of Applied Sciences and the University of Oslo for many rewarding years as both a bachelor and master student. The opportunities they have given me are greatly appreciated.

Last, but not least, i would like to thank my family for their unconditional support, belief and interest in what i do.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Data confidentiality on physical machines

Computers have revolutionized society by being able to efficiently handle (store, arrange, calculate and manipulate) information at a pace humans are incapable of competing with. The first general-purpose computers were only available for a selected few at larger organizations due to their physical size and price. Technical advances and mass production made computers cheaper to produce and by the late 1970s computers were adopted by an increasing number of households and institutions. The computers were still seldom personal and it became evident that the information users stored had to be protected from the other users using the system.

Access control lists were implemented to protect information from unauthorized access by storing the access rights for files as metadata. The weakness of this method is that the operating system evaluates the access rights runtime, meaning that the protection mechanism can be evaded by connecting the media to a machine with software that does not evaluate anything. A technology that helps solve this issue is full disk encryption. It enables administrators to encrypt the content stored to the media at run time and in order to connect the media to another machine the encryption key is needed. The weakness with full disk encryption is that the encryption key resides in memory when it is in use and it has been shown that it is possible to extract the key from memory [**1**]. The memory from a physical machine can be extracted with software such as DD or with hardware such as FireWire [**2**].

In the same way computers revolutionized information handling the Internet revolutionized information availability and exchange. First with relatively simple protocols such as HTTP and SMTP that dealt mostly with the exchange of static text and images. Later dynamic content came to market and made it possible to provide cloud services such as online banking. At this point the technology to secure stored information was no longer sufficient to keep information private, as the information could be read in transmission over the network. The answer to these challenges was to in-

troduce end-to-end SSL encryption and protocols such as IPSec.

1.2 Data confidentiality on virtual machines

Just like computers and the Internet were revolutions of the past cloud computing is revolutionizing the market today. Cloud computing is based on virtualization, a software abstraction layer that allows system administrators to consolidate several operating system instances, called virtual machines, on one physical machine, called a hypervisor or VMM. This gives several benefits over running physical machines such as: better hardware utilization, better portability and ease of management. What cloud computing adds to virtualization is an interface that can be used to create virtual machines on demand, typically over the network. Since cloud computing combines the information handling and exchange in one product it can be seen as an aggregation of a computer and the Internet. The benefits of this technology are enormous as customers now can rent computational resources from cloud providers when the need arises and stop when the resources are no longer needed. This provides a very dynamic environment and can help save a significant amount of hardware and infrastructure costs.

While cloud computing have many benefits it is important to keep in mind that it also introduces a whole new type of security challenges, while still also being subject to the security challenges that exists on physical computers as previously discussed. In general cloud computing introduces three new security challenges [**3**]:

- Information can leak between virtual machines.
- A virtual machine could compromise the VMM and in turn get access to other virtual machines.
- Information can leak between the virtual machine and the VMM.

This thesis aims at creating a prototype that can show, in practice, that information that leaks between the virtual machines and the VMM can be problematic. When dealing with virtualization in a company environment the aforementioned challenge might not seem that important, given that employees are trusted. With the introduction of public cloud computing services we are no longer dealing with a company environment however and the challenge becomes very relevant. Customers that rent computational resources have no control over how the rented resources are managed, who manages them and sometimes not even in which country they are located.

Information leak between the virtual machine and VMM is a big topic and as such it has been important to try to narrow down the scope. Since the purpose of cloud computing is to be able to get access to computational

resources on demand, it is logical to assume that whatever those resources compute will be exchanged with someone. In a physical environment the exchange of data is protected by encrypting the network traffic, which should be relatively safe given that the physical machine itself is protected. In a virtual environment it is also possible to encrypt both the network traffic and the media the files are stored on. There is however an important difference between the two (physical and virtual), namely that the virtual machine is running as an instance on a physical machine. This means that the administrator of the physical machine has full access rights to the resources of the virtual machine. As previously discussed it has been shown that it is possible to extract encryption keys used in full disk encryption from the memory of physical machines. Taking this into account it could be possible that the administrator of the physical machine the virtual instance is running on can extract files that are part of a file transfer directly from memory. Not only that, but since files are dealt with unencrypted in memory the administrator should be able to extract the files even if network encryption and full disk encryption is implemented on the virtual machine.

It is important to note that being able to extract files from memory is nothing new and has been done for quite some time in computer forensics on physical machines. There are however some big difference between being able to do this on a physical machine and a virtual. The biggest difference might be that extracting memory from a physical machine typically requires the use of additional hardware. This is both time consuming, requires the examiner to physically be at the location of the machine and might lead to the owner of the physical machine to notice that something is happening. In a virtual environment additional hardware is not needed, memory can be extracted transparently for the owner and it is easy to scale as the examiner do not have to be at the same physical location.

1.3 Problem statement

The purpose of this thesis is to show that information that leaks between a virtual machine and the VMM can be problematic. This is especially problematic now that cloud computing has gained momentum and the infrastructure the virtual machines are running on are administrated by a third party. Since the computational resources are rented over a network it is logical to assume that the data that is computed will be sent back over the network. With that in mind the focus of this thesis has been to see if it is possible, by creating a prototype, to extract the files that are part of a file transfer on a virtual machine from its memory even if the files are encrypted on the virtual machine.

The problem statement of the thesis is the following:

How can data that is secured with encryption on a virtual machine be captured by the VMM?

1.4 Thesis outline

Below follows the structure of the thesis:

- Chapter 1: Presents the motivation and problem statement of the thesis. A short introduction covering file confidentiality challenges on physical and virtual machines is given.
- Chapter 2: Presents background information that is relevant to the thesis.
- Chapter 3: Present the approach and methodology used to create, confirm and measure the prototype.
- Chapter 4: Presents the results and analysis of the confirmation and measurements of the prototype.
- Chapter 5: Presents a discussion about the findings.
- Chapter 6: Presents a conclusion to the thesis and discusses further work.

Chapter 2

Background

2.1 Virtualization

Virtualization is a software abstraction layer, typically called hypervisor or virtual machine manager (VMM), which lies between the hardware and the (guest) operating system. The purpose of virtualization is to provide a platform that can share and prioritize usage of the physical hardware resources by providing emulated hardware resources controlled by the VMM to the virtual machines.

2.1.1 Advantages and disadvantages

Advantages

Using virtualization has several advantages over running the OS directly on the hardware:

- Consolidation: Several virtual machines can run on the same physical machine, providing better hardware utilization. This saves both electricity and infrastructure costs and as a positive consequence lowers the environmental impact.
- Portability: Migration of virtual machines between physical servers is straight-forward as the hardware is virtualized and thus equal on both machines/ends. This gives the administrator great flexibility to move virtual machines around should the need arise. Such flexibility could be needed when an administrator sees that a physical host has a very high load. In that event the administrator could move one or several virtual machines to a host that has less load, to better spread the computational load over the infrastructure.
- Availability: The great portability of virtual machines can be used to increase the availability of the virtual machines. A powerful feature of virtualization is high-availability, where two physical hosts are set up, one running the virtual machines and one keeping a runtime copy of the virtual machines. In the event that the host running the virtual machines goes down, the host with the runtime copy can take

over with little to no downtime. Similarly, when an administrator needs to conduct maintenance on a physical machine the virtual machines hosted on the machine can be moved to a stand-by machine to decrease the downtime.

- Isolation: The virtual machines are isolated from each other, meaning that if one host gets compromised it does not affect the other. This is beneficial from a security point of view in several ways. This feature can be used to isolate services by setting up one virtual machine per service and in that way the compromise of one service would not affect the other services running.
- Manageability: Virtualization vendors usually provide software that allows the administrators to manage the virtual machines and infrastructure using a graphical user interface. This software is typically capable of creating, modifying and deleting virtual machines on the fly. Additionally some of the software provides APIs that can be used by administrators to automate tasks.

Disadvantages

Unfortunately there are also some disadvantages with virtualization:

- Single point of failure: The physical server hosting the virtual machines will be a single point of failure unless it is set-up in a high availability configuration. This means that great attention should be given to avoid having several critical virtual servers hosted on the same physical machine.
- Attack target: The physical machines hosting virtual machines can become attack targets just due to the fact that it is hosting several virtual machines. For attackers this is an efficient way of attacking as they can take down several servers by just attacking one.
- Shared resources: While sharing resources have a lot of benefits (eg. consolidation) it also has some disadvantages. If the resource usage of the virtual machines are not monitored and configured correctly a few virtual machines could consume all the available resources, greatly affecting the other virtual machines running on the same physical machine. Additionally shared resources can lead to several different privacy issues.

2.1.2 Types of virtualization

There are in general two types of virtualization: full- and paravirtualization.

Full virtualization

In a fully virtualized environment all the hardware is fully emulated by the VMM, which provides a total decoupling from the underlying

physical hardware. The advantage of such an environment is that the virtual machine's operating system can run unmodified, fully unaware that it is virtualized. In some situations this is necessary in order to support old legacy operating systems that were not designed to run in virtualized environments. The disadvantage of running in a fully virtualized environment is the computational overhead all the low-level emulation introduces.

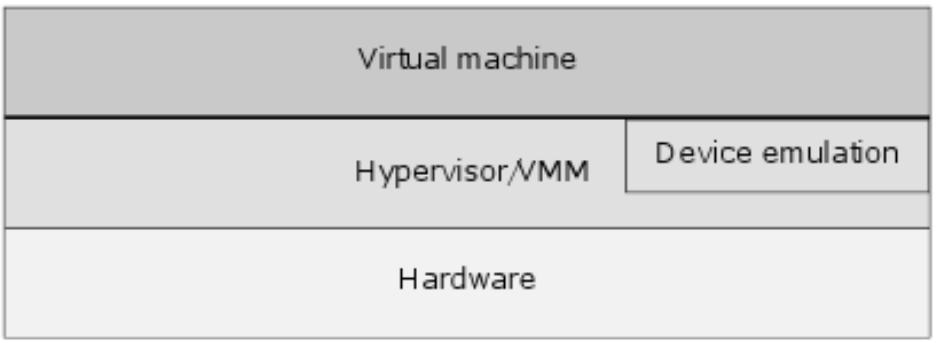

Figure 2.1: Full virtualization architecture

Paravirtualization

In contrast to a fully virtualized environment the virtual machine's operating system in a para virtualized environment is aware that it is running in a virtualized environment. This allows the virtual machine's operating system and the VMM to cooperate (via device drivers) and thus reduce the overhead of the low-level emulation. This increases performance but requires that the virtual machine's operating system has built-in support for this, something many old legacy operating systems do not.

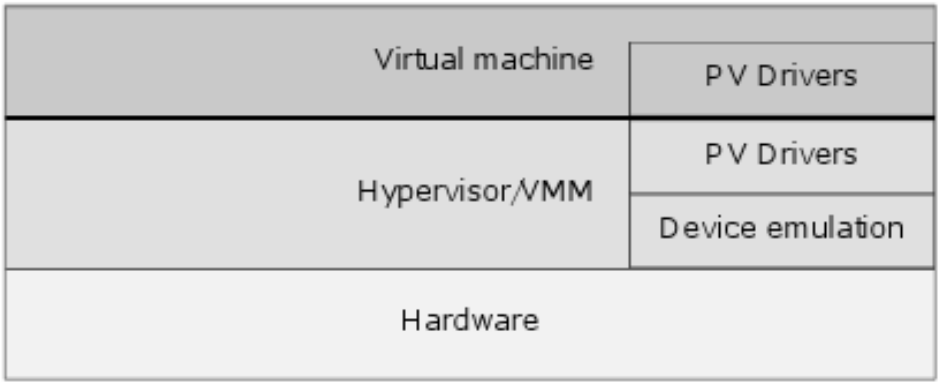

Figure 2.2: Paravirtualization architecture

2.1.3 Xen

Xen is an open source hypervisor that was originally developed as a research project at the University of Cambridge. The first versions of Xen only had built-in support for paravirtualization, but support for full virtualization has been added in later versions. Xen was the de facto standard virtualization technology used on the Linux platform and as such the hypervisor is used by several of the largest cloud computing providers such as Amazon, Rackspace and Softlayer [**4**]. This position has changed in recent years due to the introduction of KVM.

2.2 Cloud computing

Cloud computing is used to provide computational resources on demand. The definition of cloud computing from NIST defines the concept well:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [**5**]

2.2.1 Service models

Cloud providers offer several service models, depending on how much of the infrastructure they maintain for the consumer. The models can often be seen visualized as a stack, where the lowest layer provides general infrastructure (typically storage, network and hardware) and the highest layer provides a running application that can be used directly by the consumer. The stack, from bottom to top, is visualized and described below:

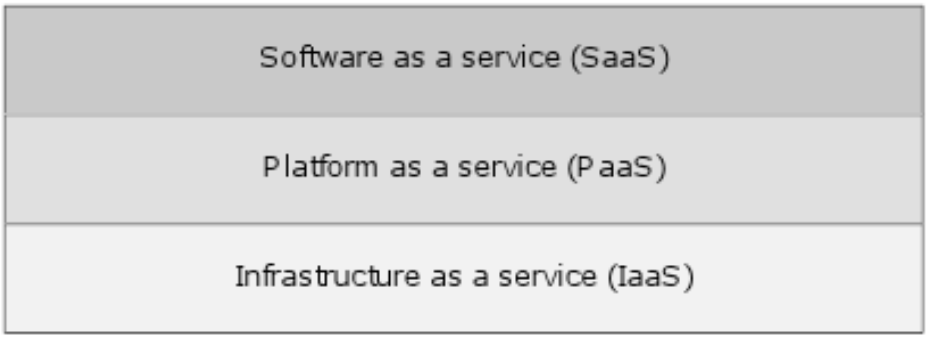

Figure 2.3: The cloud stack

Infrastructure as a service

This is the lowest layer of the stack, which provides the consumer with the ability to provision basic infrastructure such as storage, network and processing power. The basic infrastructure is maintained by the cloud provider while the software, including operating system that is running on top of the infrastructure has to be maintained by the consumer. This layer can also be seen as the virtualization layer as it is providing virtualization as an on-demand service to consumers.

Platform as a service

This is the middle layer of the stack, which provides the consumer with the ability to provision a platform that consist of the basic infrastructure, an operating system and tools that can be used to develop and run applications on the platform. The platform is maintained by the cloud provider while the applications are maintained by the consumer.

Software as a service

This is the top layer of the stack, which provides the consumer with access to use an application through a client. On this layer all the infrastructure is maintained by the cloud provider but the consumer might be able to configure the application to some degree.

An example of SaaS is Facebook, an online social network service that is accessible to consumers through a web browser. Facebook maintains both the application and the platform, but the consumer is able to change some preferences.

2.2.2 Deployment models

Clouds can be deployed based on several different models. Some of the most popular models are described below:

- Public cloud: This is probably the cloud model most are familiar with. This model consists of a cloud provider selling cloud services to the public, typically at a large scale.

- Private cloud: In this model the cloud is owned or leased by an organization privately. This is typically used to assure that the infrastructure is private and not mixed with other customers/clients.

- Hybrid cloud: A combination of the two aforementioned models. There can be many reasons for choosing a hybrid model, such as wanting to keep sensitive data within the private cloud while having less sensitive data in the public cloud.

2.3 Memory and caches

In order to use memory efficient Linux utilizes unused memory to cache data. These caches are used to transparently store data that has been or is in the process of being processed, to the memory (RAM).This enables subsequent requests to the same data to be served faster, as accessing the memory is faster than accessing the back-end storage (typically a hard disk). Most of this cached data can be freed instantly if a process is in need of more memory.

2.3.1 General

Read requests

Whenever the system gets a read request it will look for the data in the cache. If the data is contained in the cache, the request will be served directly, which greatly improves the performance. If the requested data is not there, it will either be computed directly and put into the cache, or read into the cache from a slower storage medium (typically a hard disk). While the data is written to the cache it is similarly streamed to the process that sent the read request to make the process both efficient and transparent.

Write requests

Write requests are also cached to increase performance. Generally there are two ways to cache write requests: write-through and write-back.

- Write-through: The data is written synchronously to the memory and back-end storage (typically a hard disk). This is the slowest option as the storage is typically slow and all writes have to be written to the storage at the time they occur. It however ensures that the data is always written to the persistent back-end storage.

- Write-back: The data is first written to the cache and then at a later time to the storage. At which time this occurs depends largely on the implementation. This is more efficient, but can lead to the loss of data if the system unexpectedly shuts down and the data have not been synced to disk.

2.3.2 Linux specific

Caches

Linux has several memory caches [**6**], but the two most known is probably the buffer and page cache:

- The buffer cache: The buffer cache contains file system metadata such as inode tables, direct and indirect blocks, journals and superblocks.

- The page cache: The page cache contains files stored on the file system such as: regular files, pipes, FIFOs and the files in the virtual proc file system.

Freeing caches

Due to the fact that Linux utilizes free memory to cache data, these caches have to be freed when a process requires more memory. To accommodate this Linux holds two lists; one with active memory pages and one with inactive memory pages.

- Active list: Lists all the pages that the kernel has determined is in active use.
- Inactive list: Lists all the pages that the kernel has determined to not be in active use. These pages are further divided into two categories: dirty and clean.
  - Dirty pages contain data that has not yet been written to disk and as such the content must be written to disk before the page can be freed.
  - Clean pages contain data that is already stored to persistent storage and can be freed immediately if a process requires more memory.

2.4 Memory forensics - a branch of digital forensics

As the price of commodity hardware has constantly decreased and become more affordable over the past decades both the personal and business use of computers has increased massively. This drastic adoption is often referred to as the digital revolution and many users now use their computers for many of their daily activities, leaving behind a lot of potentially important data.

This drastic change has led to an increased focus on the forensic branch of digital forensics and the creation of specialized groups within law enforcement to deal with computer related investigations. The FBI was one of the first agencies to create such a group when they created the Computer Analysis and Response Team in 1984 [**7**].

2.4.1 Methods of investigation

Dead and live analysis are the two general methods of investigation in computer forensics [**8**] as described below:

Dead analysis

Dead analysis is the traditional way of investigation in computer forensics and is committed when the computer is shut off. It is done by making

an exact copy of the computers storage media, typically using a blocking device to ensure that no data is written to the original storage. The computer is then archived as evidence and the analysis committed on the copy in order to preserve the integrity of the original evidence.

The drawback of this kind of investigation is that only persistent data can be analyzed and therefore a lot of potentially important data stored in volatile media such as RAM and caches are ignored. This is a particularly problematic method if the disks are encrypted, as the investigators will not be able to collect any data.

Live analysis

Live analysis is investigation done on a running system. The advantage of doing an analysis on a live system is that the investigator can analyze volatile data which is only present when the system is running. Such media can contain a lot of interesting information such as open network connections, logged in users, open files and in some instances encryption keys.

The drawback of live analysis is that it is challenging one of the main concepts of computer forensics, which is that no changes should be done to the system/evidence. This is important because changing the system might lead to the court questioning the evidence and also loss of actual evidence. Evidence can be lost because the investigator typically has to import binaries in order to do the investigation. By importing binaries they might overwrite previously deleted data on disk and make caches drop data in memory when the binaries are ran and require memory.

2.4.2 Volatility

Volatility is a set of open source tools that enables users to analyze memory dumps from a large range of 32 and 64-bit operating systems, including Windows, Mac, Linux and Android systems. The tools consists of several plugins that work by iterating memory structures to provide the user with high level information such as open files, network connections and process lists. The tools are intended to both introduce people to the techniques of memory analysis and to provide a platform for further research into the area [**9**].

Volatility is shipped with several premade plugins, each with their own specific features and purpose, which lets the user analyze the memory for specific information. An example of such a plugin is the *linux_lsof* plugin, which analyses a given memory dump and lists all the open files on the system. A great feature of Volatility the extensible and scriptable API, which allows, among other things, users to create their own plugins if the provided plugins does not fit their needs.

2.5 Virtual Machine Introspection

Virtual machine introspection is a technique used to monitor virtual machines through the VMM or another privileged virtual machine. The goal of virtual machine introspection is to be able to monitor a virtual machines state outside the virtual machine itself. This allows the administrator to extract information from the virtual machine without affecting its operation or changing its state. The information that can be extracted is typically processor registers, memory, disk, network and other hardware-level events.

The term was first used by Garfinkel and Rosenblum [**10**]where they described the benefits virtual machine introspection would give in the intrusion detection field. They argued that host based intrusion detection is able to get a good view of what is happening in the hosts software, but is highly susceptible to attack. Network based intrusion detection on the other hand is more resistant to attack but has a poorer overview of what happens on the host. By combining virtual machine introspection with IDS one could both get a good view of what is happening on the host and be resistant to attack, due to the fact that the monitoring is happening outside of the virtual machine itself.

2.5.1 VMI Tools

VMI Tools, a further development of XenAccess [**11**], s a set of open source tools that enables VMI on the KVM and Xen virtualization platforms. The tools are designed to run on Linux and Mac OSX, with OSX having limited functionality.

At the heart of VMI tools lays libvmi, a C library which main focus is to read and write data from the memory of virtual machines. Additional functionality includes functions for accessing CPU registers, memory events, pausing and unpausing a VM and reading memory snapshots saved to file. Support for virtualization platforms is implemented through "drivers" and libvmi is setup to dynamically determine which virtualization platform that is present on the system at startup. This makes the library extensible beyond the scope of just the KVM and Xen virtualization platforms. This is a great feature for VMI developers as applications only have to be developed once to support several virtualization platforms and the focus of the developers can remain on VMI features rather than platform support [**12**].

In addition to libvmi the VMI Tools package include the following:

- pyvmi: A feature complete Python wrapper for libvmi, allowing developers to develop VMI applications using Python.

- pyvmi address space: A volatility address space plugin that enables volatility to analyze the running virtual machine's memory directly with its rich features.

- pyvmifs: a VMI application using the Pyvmi API that enables the user to mount the memory of a running virtual machine as a FUSE file system. This is useful as several VMI and forensic tools, including volatility, are designed to use images (regular files) as the data source.

Chapter 3

Approach and methodology

3.1 Creating the prototype

3.1.1 Software selection

In order to create the prototype three important pieces of software were needed:

- VMM software to host the virtual machines.
- Virtual Machine Introspection software to export the memory of the virtual machines for further analysis.
- Memory analysis software with the capability to find established network connections, open files associated the connection and the ability to dump found files if present in memory.

Fortunately there exist several software solutions that can provide this kind of functionality and in order to choose which software solution to use a list of requirements was compiled, as described below.

VMM

Only one requirement was set for the VMM, which was that it should be in wide-spread use to provide public cloud computing instances. This requirement was set to assure that the research put into creating and analyzing the prototype was worth-while and relevant.

Xen was chosen as the VMM to use because it has been the de facto standard hypervisor on the Linux platform and consequently is used by a wide range of the largest cloud providers such as Amazon, Rackspace and Softlayer [**4**].

Virtual machine introspection

Since Xen was chosen as the VMM the virtual machine introspection software had to be compatible with it. Additionally it would be nice if the virtual machine introspection software did not modify the VMM itself, both

to assure that it did not affect the code base and operation of the VMM (an important property of TCB) and to ensure transparent export of the virtual machine's memory. The latter requirement is set to show that memory export and analysis can be done without any of the guests/customers noticing that it is happening (due to restarts or other events that otherwise might make them suspicious).

VMI tools were selected as it fulfills all the requirements and because there are no other virtual machine introspection that fulfills them.

Memory analysis

Since VMI tools was selected as the virtual machine introspection software the memory analysis tool had to be compatible with it. Additionally, as previously mentioned, the tool had to be able to do the following: find established network connections, open files associated the connection and the ability to dump found files if present in memory.

Volatility was selected as the memory analysis tool. The reason for that was that VMI tools has an address space plugin for volatility, which simplifies the analysis of memory. Additionally Volatility has great documentation online and is able to do the analysis that is required.

3.1.2 Review of software functionality

The purpose of this subsection is to review the software functionality that will be used in the prototype.

VMI tools

VMI-tools is able to export the memory of a virtual machine to the host in two ways:

- Using the pyvmi address space, which is a Volatility address space plugin that enables Volatility to analyze the running virtual machine's memory directly.

- Using pyvmifs, which is a VMI application that uses the pyvmi API to enable the user to mount the virtual machine's memory as a FUSE file-system on the host.

Since Volatility is used to analyze the memory in the prototype, both of the aforementioned methods could be used. In order to select the most efficient method one of the Volatility commands used in the prototype, *linux_lsof*, was measured by recording the execution time for 100 runs/trials using each of the aforementioned methods. The results can be seen in the histogram below:

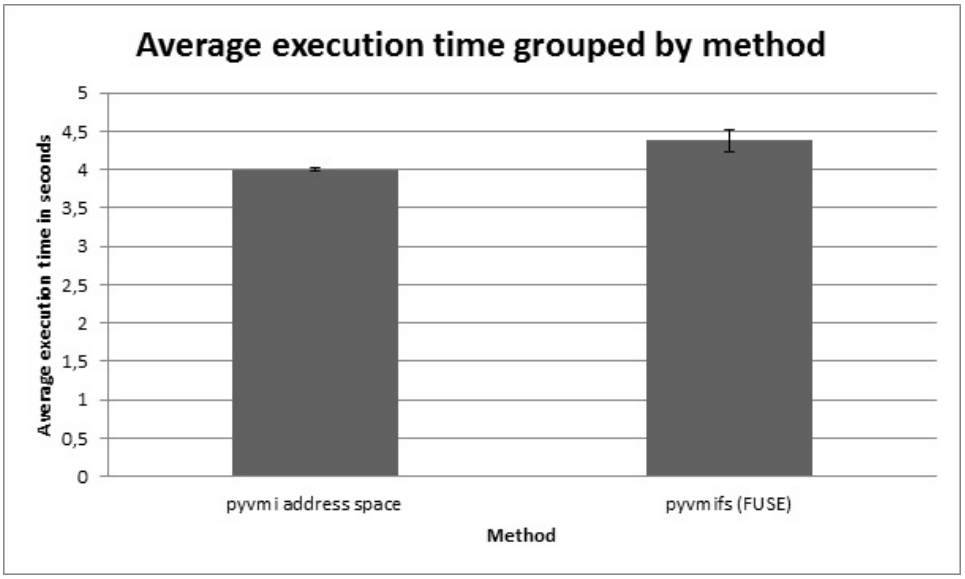

Figure 3.1: Comparing pyvmi address space and pyvmifs

The error bars shows one standard deviation away from the mean, calculated using the sample standard deviation. The histogram shows that the pyvmi address space is the most efficient method as the *linux_lsof* command takes less time to execute on average when using this method. Additionally the standard deviation is narrower, which means that the execution time does not fluctuate that much, which is important in order to get stable results from the prototype.

Volatility

Volatility is used to analyze the virtual machine's memory. Volatility is not capable of extracting the virtual machine's ongoing file transfers directly, but this can be achieved by running several Volatility commands in sequence. The sequence of commands and the information each command provides is described below:

linux_netstat

Lists the network connections on the target system with the source IP and port, destination IP and port, connection state and the process ID of the process that is associated with the connection. By filtering the output to only display *ESTABLISHED* connections, only the connections that might have ongoing file transfers are returned.

linux_lsof

Lists all the open files on the system, including the *PID* of the process that opened the file. The output of this command must be filtered to only display the files opened by processes (PIDs) that are associated with

ESTABLISHED connections captured by the previous command. Some processes have additional files open such as log files, in that case the prototype has to filter out those files using application specific filters.

linux_find_file

This command must be used in two steps:

1. First the *inode* of the given file must be found in order to be able to dump it. The file names given to this command is the file names found in the previous step.

2. By providing the *inode* found in the previous step the cached file content of the *inode* can be extracted to the disk on the host that runs the prototype. This however requires that the file content is still fully present in the virtual machine's memory.

After going through the sequence the prototype will be able to log and store the following regarding the file transfer: source IP and port, destination IP and port, process id (PID) and application name of the application transfering the file, the file name and file content of the transferred file.

3.1.3 Prototype implementation

The purpose of this subsection is to review the actual implementation of the prototype.

Programming language

Perl was chosen as the programming language to develop the prototype in. There were several reasons for this:

- Perl is the programming language that the author is the most familiar with.

- Perl is supported on a number of platforms, including Windows, Unix, Linux and Mac.

- Perl is an interpreted programming language which makes creating prototypes efficient as you do not have to compile between each code-test iteration.

- Perl has great support for text manipulation which is a great advantage as the output of the Volatility commands are mostly text.

Actual implementation

This section covers the actual implementation of the prototype. As described in the previous section three Volatility commands have to be run in sequence to be able to extract a virtual machine's file transfers from memory. To modularize and ease the complexity of developing the

prototype, the handling of each command has been implemented as its own module. To store and share data between the modules a database with three tables has been set-up. The design of the database is shown below and the description of how the fields are used will follow in the next sections where each module is explained.

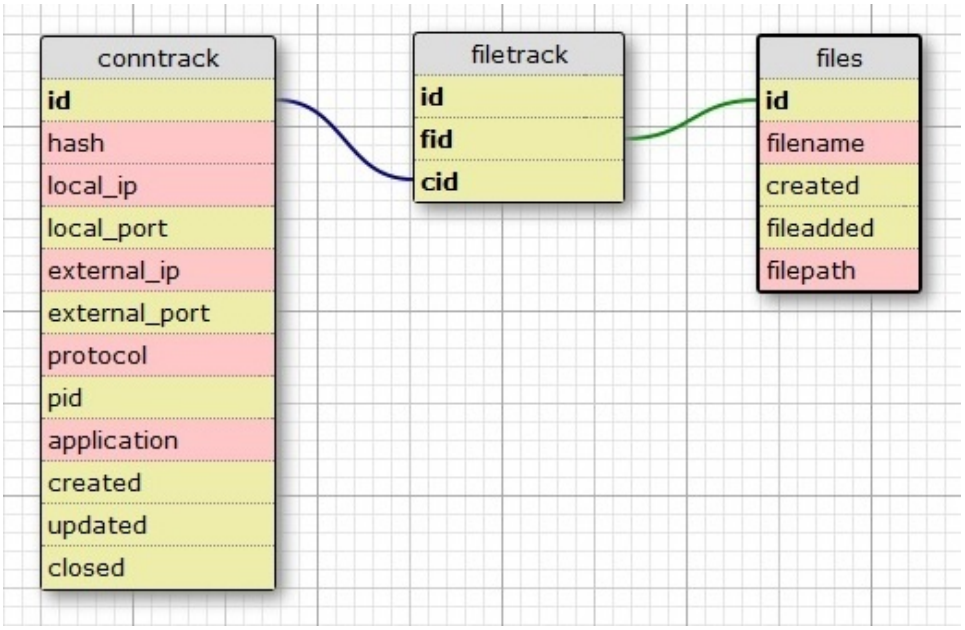

Figure 3.2: Prototype database design

The conntrack module

The purpose of the conntrack module is to enable the prototype to track established connections as they might have ongoing file transfers. This is done by storing an entry to the conntrack table for each unique established connection that is found (using the *linux_netstat* Volatility command). Each entry in the table consists of the following:

- id: An auto generated incremental id.
- hash: A MD5 hash that is generated by using the connection string retrieved from the *linux_netstat* command as input.
- local_ip and local_port : The virtual machine's IP and port that is part of the connection.
- external_ip and external_port: The external IP and port the virtual machine is communicating with.
- protocol: The protocol the communication is over, usually TCP or UDP.
- pid and application: The id and name of the process on the VM that is associated with the connection.

- created: A unix-timestamp describing when the connection was first found.
- updated: A unix-timestamp describing when the connection was last seen/updated.
- closed: A unix-timestamp that is set when the connection is no longer present.

When the module starts it will query the database for all established connections that has the *closed* attribute set to *NULL*. The hash value of those connections is then put into an array named *%openConnections*.

After this an infinite loop is started, and each iteration does the following:

1. Sets the *$execStartTime* variable to the current unix-timestamp.
2. Runs the *linux_netstat* Volatility command and filter the result to only contain established connections.
3. Loops each entry in the result and generates an MD5 hash of the entry. It then checks if the MD5 hash is present in *%openConnections* (ie. If the connection is present in the database already or not).
    - If it is present, the *updated* value for the entry in the database is updated with the current unix-timestamp. Additionally the *hash* is updated in *%openConnections* to have the value of *$execStartTime*.
    - If it is not present, an entry is added to the database with all values set except for *closed*. Additionally the *hash* is updated in the *%openConnections* to have the value of *$execStartTime*.
4. Loops each entry in *%openConnections* and checks whether or not the current entry has been updated in this iteration (by checking if the value is *$execStartTime* or not). If the entry has not been updated the *closed* value is set to the current unix-timestamp in the database and the entry is removed from *%openConnections*.

The filetrack module

The purpose of the filetrack module is to track the files opened by processes that are associated with currently established connections. It is important to note that a process can be associated with several connections, based on whether the process forks or not when it receives a new connection. For this reason the filetrack module adds entries to two tables as described below:

files: This table contains the information about the actual files.

- id: An auto generated incremental id.

- filename: The path to the file on the target system (virtual machine). In retrospect this value should have been renamed to something more descriptive.
- created: A unix-timestamp describing when the file was found.
- fileadded: A unix-timestamp describing when the file was extracted from the target system's (virtual machine) memory.
- Filepath: The path to the extracted file on the local file system (the file system of the host running the prototype).

Filetrack: This is a help table that maps files to connections due to the fact that files can be associated with several connections.

- id: An auto generated incremental id.
- fid: A foreign key referring to the file in the *files* table.
- cid: A foreign key referring to the connection in the *connection* table.

When the module starts it starts an infinite loop that for each iteration does the following:

1. Queries the database table *conntrack* for connections where the value of *closed* is set to *NULL*. This is done because we only want to track files of currently established connections.
2. Maps process id's to connection id's in an array called *%pidCid*.
3. Maps the process id's to the process name in the *%pidApp* array, eg. 3242 to Apache.
4. Runs the *linux_lsof* Volatility command to find the open files on the system and filters the output to only contain files opened by processes present in the *%pidCid* array.
5. Loops through the processes in *%pidCid* and does the following for each iteration:
   (a) Further filter the result of the open files to only contain files opened by the current process.
   (b) Runs an application specific filter if such a filter is defined. These application specific filters are useful when dealing with applications that have files open that are not associated with file transfers. Apache for example usually has some log files opened, which we are not interested in.
   (c) Adds the remaining files of the results to an array called *%pidFiles*.

(d) Loops through all the connections associated with the given process and runs a query that returns the files that are already associated with the connection in the database. The *id* of the connection is added to the files in *%pidFiles* that are not already associated with the connection.

(e) *%pidFiles* are looped and the files that have references to connections are added to the *files* table. The file and connection(s) are then connected by adding entries to the *filetrack* table.

The filedump module

The purpose of the filedump module is to dump the files that were registered by the filetrack module and that are still part of established connections (the *closed* status of the connection in the *conntrack* table is *NULL*). The filedump module updates the *fileadded* and *filepath* values in the *file* table if it manages to extract the file.

When the module runs it starts an infinite loop that for each iteration does the following:

1. Queries the database for files that are associated with established connections and are not yet extracted. The *filename* with the associated *id* is added to an array called *%openFiles*.

2. Loops through the files in *%openFiles* and tries to extract the files from the memory of the virtual machine and store them to the disk of the machine running the prototype. If it manages to store the file to disk the *fileadded* and *filepath* values are updated in the *files* database entry.

3.2 Verifying the prototype and measuring important properties

3.2.1 Creating the workload

In order to both verify the prototype and measure other important properties a workload has to be created. Selecting a workload that is representable for file transfer usage in the cloud is not a trivial task, as the cloud can be used for almost any purpose. One way to get an idea of a common usage is to look at what kind of cloud services the global internet users access often. There are several internet sites that rank the most accessed web-pages, one of the major ones being Alexa. By looking at the 50 most visited webpages [**13**] several SaaS cloud services are listed: Facebook, Youtube, Twitter, Linkedin, Blogspot, Wordpress, Instagram, Tumblr and Imgur. Common for most of these services when it comes to file transfers is the ability for the users to share (upload) and view (download) pictures.

To create the workload 200 pictures were collected in total from four of

the previously mentioned SaaS sites, 50 pictures from each site. The selected SaaS sites were: Facebook, Blogspot, Instagram and Tumblr. The reason for selecting those four was simply that they all have gallery functionality which makes the collection of images more time efficient.

In order to run the workload a server serving the pictures and a client browsing/downloading the pictures has to be set-up:

The server

Setting up a server that is capable of serving static image files is a trivial task. All that is needed is to transfer the payload (pictures) to the server and install a HTTPD-server that can serve them.

However, to show that files can be extracted from memory even if general virtualization security principles are followed, the Red Hat virtualization security guide was consulted [**14**]. The reason for consulting only this guideline was that many of the available guidelines were either very theoretical [**15**] [**16**] or deal with more general system administration principles [**17**]. The Red Hat virtualization security guideline is rather comprehensive and contains a lot of principles that are outside the scope of file confidentiality. Since this thesis deals with file confidentiality, only the principles that deals directly with that has been taken into account. The Red Hat virtualization security guide states the following in regard to file confidentiality:

- "In virtualized environments there is a greater risk of sensitive data being accessed outside the protection boundaries of the guest system. Protect stored sensitive data using encryption tools such as dm-crypt and GnuPG; although special care needs to be taken to ensure the confidentiality of the encryption keys."
- "Ensure that guest applications transferring sensitive data do so over secured network channels. If protocols such as TLS or SSL are not available, consider using one like IPsec."

Taking this into account the server has to be set-up with an encrypted file system and a HTTPD-server that is able to serve requests over SSL.

The client

There exist several tools that are capable of generating a HTTP request workload. Since the author is already familiar with Httperf [**18**] that tool was selected.

When it comes to generating the actual workload Httperf has a number of options that can be used to generate different types of workloads. Since the payload consists of gallery pictures a natural workload would be that of a user browsing the galleries. Assuming that the user watches every picture for approx. 1 second a session workload can be setup with Httperf. The session workload will create one connection to the server and

request one picture every second until it has requested all the pictures. The Httperf command that was used was the following: *httperf –server <ip> –wsesslog 1,1,workload –max-connections=1 –ssl* The workload file that was used is provided in Appendix F on page 69

3.2.2 Verifying the prototype

The purpose of the prototype is to show that it is possible to extract files that are part of an ongoing file transfer from the target systems memory and log important information regarding the file transfer. To verify the prototype the workload has to be run against a server and the information that is recorded, including the files extracted, has to be verified. Since the prototype stores all information regarding the file transfers in a database a query can be used to display the recorded information, including the location where the extracted files were put. The information that is recorded can easily be verified because the information about the client requesting the workload is known. The extracted files can be verified by using a hashing function that will take the file as input and produce a fixed-length string that uniquely identifies the file. By comparing the hashes of the files that were extracted with the hashes of the same files on the server it can be verified that they are in fact the same.

An important factor to take into account when running the workload is the network speed between the client and the server as it will affect the number of files that the prototype will be able to extract. This factor will be discussed more thoroughly in later sections, but for the purpose of verifying the prototype the speed between the client and the server was set to 100 kB/s.

3.2.3 Measuring the coherence between different memory configurations and the execution time of the analysis commands

An interesting property to measure is to what extent the total memory configuration of the target virtual machine will affect the execution time of the memory analysis commands used in the prototype. As the prototype is constantly analyzing the memory, the time this takes will greatly affect the number of files that can be extracted from memory (the more times the analysis commands can run the more files it will find).

To test this each of the analysis commands (*linux_netstat*, *linux_lsof*, *linux_find_file*) can be ran 100 times against different memory configurations and the execution time can be logged. Due to time limitations the number of memory configurations has to be limited, which makes it important to select memory configurations that will easily show if there is significant difference between the execution times. One way to do that is to start at a low memory configuration, make the next configuration the double of the previous and so on. By testing against four such configurations the highest configuration will be eight times larger than the smallest.

This should show a significant difference if there is coherence between the memory configuration and execution time. The memory configurations used in these tests were 256 MiB, 512 MiB, 1024 MiB and 2048 MiB.

The reason for running the Volatility commands and to not use the prototype was that the prototype does additional work (such as database inserts, updates and queries) and as such the measurements would measure more than just the actual memory analysis.

3.2.4 Measuring the coherence between the network speed and the number of extracted files

Since the purpose of the prototype is to be able to extract files associated with file transfers it is interesting to see how the network speed will affect the number of extracted files. As previously described the prototype consists of three modules, one of which deals with file tracking (finding files associated with a connection), called filetracker. This module lists all the open files on the system and then associate them with connections. To do this the Volatility command *linux_lsof* has to be executed and the open files belonging to established connections that are not yet added to the database has to be added. This leaves two open time windows:

- The time between when the *linux_lsof* command is ran and finishes.
- The time it takes the module to update the database.

With the two open time windows it is logical to assume that when dealing with small files and fast network speeds several files will never be detected, because the files will reside in memory for a very short time.

To test this the workload can be ran 30 times against each network speed. The reason for only running the workload 30 times is that it takes a long time to run the workload, especially at low network speeds. Due to time limitations the number of network speeds has to be limited as well. This makes it important to select network speeds that will easily show if there is significant difference between the number of files that can be extracted. One way to do that is to start at a low network speed, make the next network speed the double of the previous and so on. By testing against four network speeds the highest will be eight times faster than the smallest, which should show significant difference if there is coherence between the network speed and the number of extracted files. The network speeds used in these tests were 50 kB/s, 100 kB/s, 200 kB/s and 400 kB/s.

3.2.5 Measuring the coherence between the network speed and the average file size

As described in the previous section there are two open time windows that limit the number of files that can be extracted. The first time window is between each time the *linux_lsof* analysis command is ran and the second

is when the module updates the database. Because of these time windows small files will probably be hard to extract from memory, because they reside there for such a short period. To test if that assumption is correct it is interesting to see if the average file size of the extracted files increase when the network speed increase.

The measurements from 3.2.4 can be used to calculate the average size of the extracted files based on the network speed and as such no new measurements have to be made.

3.3 System specification and set-up

The purpose of this section is to provide a simple overview of the systems that were used and how they were set-up. The purpose is not to function as a tutorial, as it is outside the scope of the thesis. There exists a lot of good documentation online that will be referenced instead and the package names of the installed software will be provided.

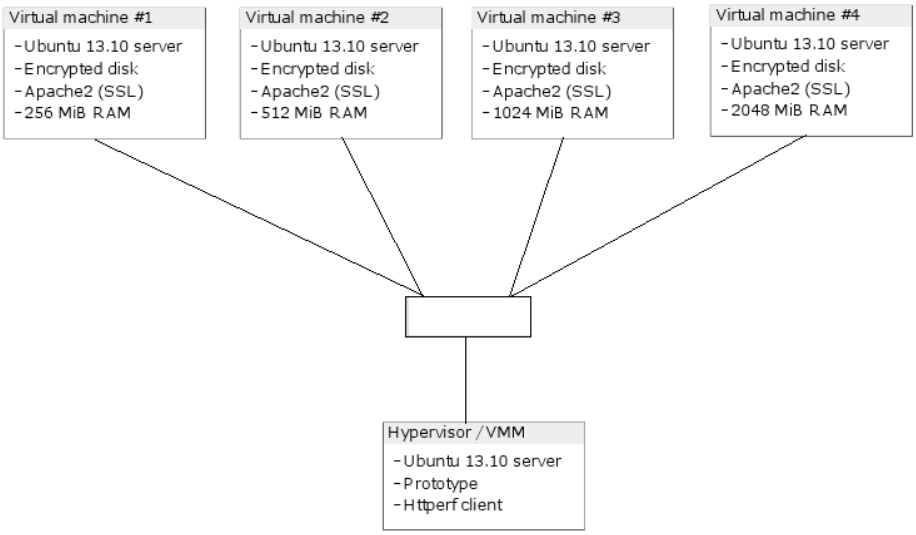

Figure 3.3: Infrastructure overview

3.3.1 Environment

The physical server including all experiments ran in a server room at Oslo and Akershus University College of Applied Sciences to take advantage of the infrastructure these kinds of environments provides such as cooling, stable electricity, good network connectivity and physical access control.

3.3.2 The physical server

The physical server was used as a VMM housing the virtual machines. Additionally it was used to run the prototype and as the client requesting the workload from the virtual machines.

Specifications

The physical server used was a Dell R710 with the following specifications:

- CPU: Two Intel Xeon L5630 processors with a total of 8 cores and 16 threads running at 2.13GHz.
- RAM: 72GB ECC DDR3.
- Disks: Two 146GB 15K RPM SAS disks connected to a Perc H700 controller. Configured in RAID0 for increased performance.
- Network: Gigabit Ethernet.

Set-up

The following software was installed:

- Ubuntu Server 13.10
- Xen: *xen-hypervisor-amd64* [**19**]
- VMI-Tools 0.10.1 [1] [**20**]
- Volatility 2.3.1 [2] [**21**]
- Perl: *libdbi-perl libdbd-sqlite3-perl* (used by the prototype).
- Httperf: *httperf*

VMI-tools was quite tricky to set-up as it had several dependencies that were not well documented. The dependencies needed are the following: *build-essential libxen-dev libtool automake bison flex check libfuse-dev libglib2.0-dev python-dev python-fuse*

3.3.3 The virtual machines

The virtual machines were used to serve the workload that was generated by the client. It was the memory of these machines that were analyzed by the prototype.

[1] http://vmitools.googlecode.com/files/libvmi-0.10.1.tar.gz
[2] https://volatility.googlecode.com/files/volatility-2.3.1.tar.gz

Specifications

- CPU: 1 CPU core.
- RAM: 256MiB-2048MiB depending on configuration.
- Disk: 8 GiB.
- Network: Gigabit Ethernet.

Set-up

To follow best practices in regards to file security in virtual environments the virtual machines were set-up with full disk encryption (selected during OS install). Additionally Apache was configured to serve requests over SSL.

Installed software:

- Ubuntu Server 13.10
- Apache: *apache2 libapache2-mod-bw* (used to limit the outgoing bandwidth for some tests) [**22**].

Creating a Volatility profile

In order for Volatility to be able to analyze the memory a Volatility profile has to be generated on one of the virtual machines and transferred to the machine that runs the prototype. These profiles describe memory structures and other important data the Volatility command needs to correctly analyze the memory [**23**].

Chapter 4

Results and analysis

The purpose of this section is to go through and analyze the results of the tests previously described in the approach chapter. A couple of mathematical theories have been central in generating the results and in order to keep repetitive content low these theories will be explained briefly below and referred to throughout the chapter.

Population and sample Populations and samples are two different types of dataset. A population refers to a whole group of objects that has something in common, such as all engineers. It can be problematic to collect data from all objects in a population and as such a sample, which is a smaller group of the objects in the population, is often used to approximate. Due to the difference of the sets the formula used to calculate the standard deviation of a sample differs from that of a population. This is because the formula has to take the uncertainties of not having all the data in the set available into account.

Standard deviation Standard deviation is a measure of how spread out the numbers in a set is. A low standard deviation describes a set with numbers that are closely concentrated, while a high standard deviation describes a set where the numbers are more spread out. A low standard deviation is typically expected when sampling static computational workloads, which is the case for several of the tests in this thesis, as the result should be almost identical between each iteration. There will be some exceptions to this however, such as occasional noise that is caused by background OS/kernel processes.

The normal distribution, empirical rule and confidence intervals The normal distribution is a very important distribution in statistics both because it is often found to be the distribution of many natural phenomena (eg. height and weight) and because it has some very useful properties. The distribution itself is symmetrical and shaped as a bell-curve.

The empirical rule, which applies to the normal distribution, states that 68% of the values of a population will be within one standard deviation,

that 95% of the values will be within two standard deviations and that 99.7% of the values will be within three standard deviations from the mean. This is useful in statistics, given that you already know the mean and standard deviation of the population. Knowing these values you can calculate the chance of getting a value within an interval, known as a confidence interval. As an example, if a population has an average of 2 and the standard deviation is 0.5 the 68% confidence interval is between 1.5 and 2.5.

In practice it can be hard to use the properties of the normal distribution as the average and standard deviation of the whole population is seldom known.

The Central limit theorem and the t-distribution The central limit theorem states that if you take samples from any population, calculate the averages and plot the distribution of averages it should become approximately normal distributed given that the sample size is large enough. How large the sample size has to be depends on the underlying population/distribution, but a good approximation is that the sample size has to be around 30 or larger.

This allows us to calculate confidence intervals based on a single sample to approximate how likely it is that the average calculated is within an interval of the true average. How the confidence intervals should be calculated depend on how much that is known about the data. There are at least three different cases as described below:

1. The distribution is normal and the variance is known.
2. The distribution is normal and the variance is unknown.
3. Both the distribution and the variance is unknown.

In the first case the confidence interval can be calculated using the confidence interval of the normal distribution. In the two latter cases the most correct way to calculate the confidence intervals is using the t-distribution. The reason for that is that the t-distribution takes the uncertainties (unknown variance and unknown variance and distribution) into account.

4.1 The developed prototype

The developed prototype consists of four scripts and an SQLite database:

- wrapper.pl: A wrapper used to run the scripts. This wrapper is using threads to enable the scripts to run at the same time in a modular fashion. The script is provided in Appendix A on page 55.
- conntrack.pl: The conntrack module that tracks all established connections found on the target virtual machine. These connections are tracked because established connections might have ongoing file transfers. The script is provided in Appendix B on page 57.
- filetrack.pl: The filetrack module that tracks open files associated with established connections (found by conntrack). This module has a filter that will filter out application specific files that might be open, such as log-files. The script is provided in Appendix C on page 61.
- filedumper.pl: The filedumper module tries to extract files found by the filetrack module from the memory of the target virtual machine to the disk of the machine running the prototype. The script is provided in Appendix D on page 65.
- The database: The database is used to store the information collected by conntrack.pl, filetrack.pl and filedumper.pl. The database is a SQLite database and the definition/create-script is provided in Appendix E on page 67.

4.1.1 Configuration

The scripts needs some configuration in order to work, such as the path to the Volatility binary and information regarding the database. To simplify the configuration is equal in all the script files and the properties that can be configured are described below:

Volatility

```
# Volatility configuration
my %volatility = (
    bin => '/root/volatility-2.3.1/vol.py',
    command => 'linux_netstat',
    file => 'vmi://ubuntu1024',
    profile => 'Linuxubuntu-13_10-serverx64'
);
```

- bin: Is the path to the Volatility binary on the machine running the prototype.
- command: Is the name of the Volatility command that the script should use to analyze the memory. This variable should not be changed.

- file: Is the path to the file/memory that the Volatility command should analyze.
- profile: Is the profile the Volatility command should use when analyzing the memory. These profiles describes memory structures and other important data the Volatility command needs to correctly analyze the memory.

Database

```
# DB configuration
my %db = (
        driver => 'SQLite',
        database => 'db.db',
        username => '',
        password => ''
    );

# DB tables configuration
my %dbTables = (
            conntrack => 'conntrack',
            files => 'files',
            filetrack => 'filetrack'
);
```

When using the database provided in the appendix the only variable that has to be changed is the database variable. This variable specifies the path to the database on the machine running the prototype.

4.2 The Workload

The workload used in the thesis consists of 200 pictures collected from four different SaaS sites. This section aims at giving the reader a better understanding of how the workload is distributed.

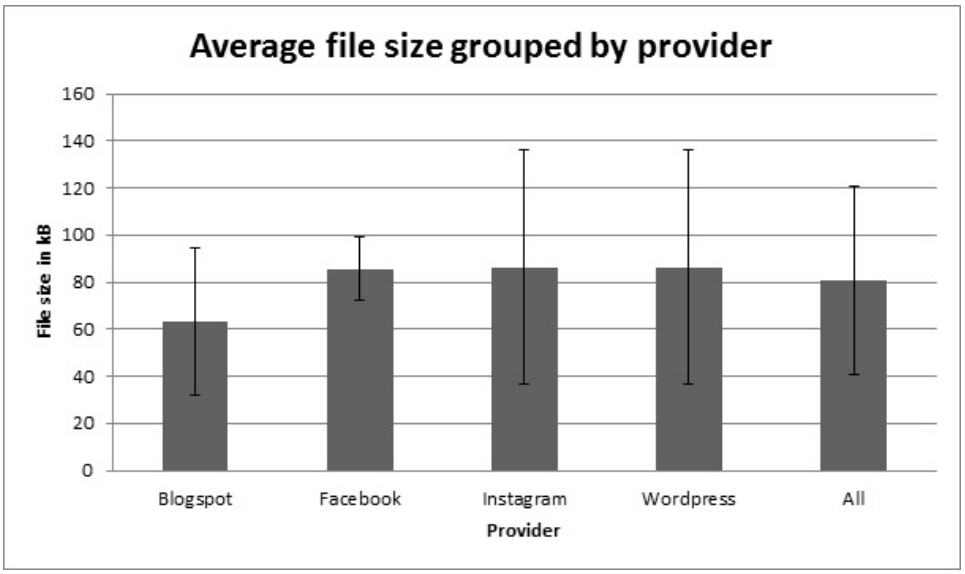

Figure 4.1: The workload's average file size grouped by provider

The histogram above displays the average image file size grouped by SaaS provider, with the error bars showing one standard deviation from the mean. The provider *all* is the set of all the image file sizes and as such represents the whole workload. The graph shows that the image file sizes differ greatly (high standard deviation). This could mean that the file sizes are evenly spread out over a larger spectrum or that most of the file sizes are concentrated with some files differing greatly in size from the norm. It could also be a combination of both. To get a better idea of how the workload is distributed a frequency file size distribution histogram is provided below.

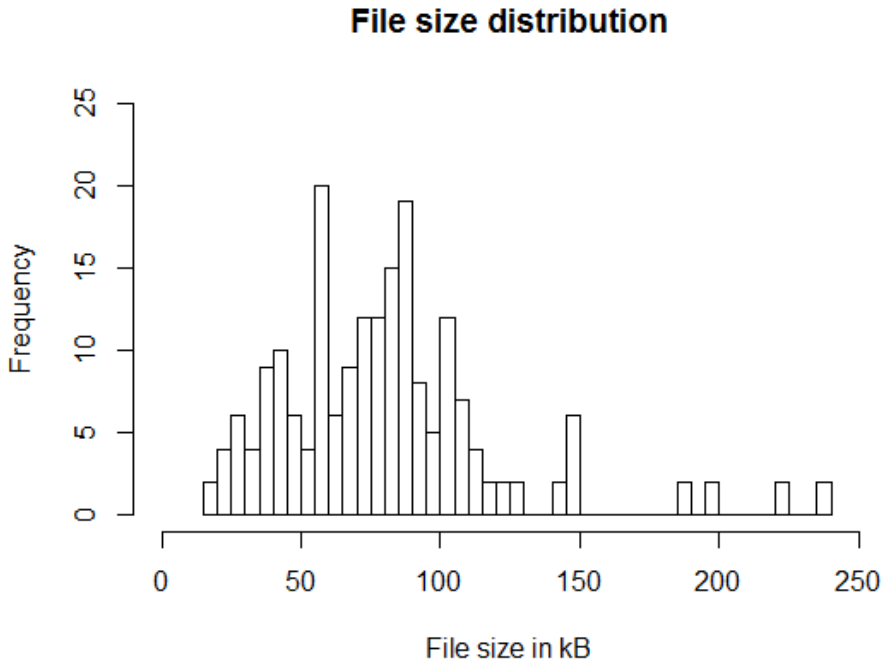

Figure 4.2: The workload's average file size frequency distribution

The histogram above shows the frequency file size distribution of the workload. The histogram shows that most of the images have a file size in the interval between approx. 20-125 kB, with just a couple of images above 200 kB. This means that the workload is skewed towards the left and that it consists of mostly smaller images with a few exceptions of larger images.

4.3 Verifying the prototype

This section aims at verifying that the prototype is working correctly. To check this an MD5 hash was generated for each of the files that were extracted and compared to the hash of the same files on the target virtual machine. Additionally a query was ran against the database to show that it managed to register correct information and that the files that were listed as extracted were actually present on the file system of the machine running the prototype.

4.3.1 Comparing the extracted files with the files on the target virtual machine

Filename	MD5 hash of file on guest	MD5 hash of extracted file
blogspot_06bffe74611bdd2b9ff872ddbce570ad.png	3a5f0ed621d4a91e71d1552c3f43d839	3a5f0ed621d4a91e71d1552c3f43d839
blogspot_468934d7f5ad6d9c7774a838404dc985.png	9666bf35aefe281680ad1772e8315075	9666bf35aefe281680ad1772e8315075
blogspot_675318a2779027329cd8ac494885c41f.png	5f0b3374207ba60eaeb67d54405cbbb2	5f0b3374207ba60eaeb67d54405cbbb2
blogspot_ff0a81287436e721541243946676c9d6.png	af0d55bd006d25f78af9c93218d900b0	af0d55bd006d25f78af9c93218d900b0
facebook_06b0f3f803bf320ba90b0b010c3068b2.jpg	dabab61e0c2dc556b562bba7cb44aea5	dabab61e0c2dc556b562bba7cb44aea5
facebook_513e5dc74b2157cf435831158eca858e.jpg	2c3ee5eba1e821c64c061a73679c8099	2c3ee5eba1e821c64c061a73679c8099
facebook_51e2663a88befc0126b12ffddb896690.jpg	4ec52c217e28079d102370583a81f1f3	4ec52c217e28079d102370583a81f1f3
facebook_53aa521e7c37e20d26ee118564b0c11c.jpg	128b22f4e7803100c19fe16bdb4a0979	128b22f4e7803100c19fe16bdb4a0979
facebook_656c2abe53056907e813d74a11aea29f.jpg	c6eb27ccec0ef100f79613ecdd2b69a5	c6eb27ccec0ef100f79613ecdd2b69a5

Figure 4.3: Comparing MD5 hashes of extracted and served files

The table above is just a sample of the collected data. The full result is provided in Appendix G on page 73. As the sample shows the MD5 hash for all of the extracted files match the MD5 hash of the same files on the target system. This means that the prototype is able to extract the files correctly.

4.3.2 Verifying registered information

Below a query is ran against the database to show the external IP, local port and the location of the extracted file on the file system the prototype ran at. The prototype registers more information than this, but in order to make the result fit the page width the output was limited. This is however sufficient to verify the most important information, which is the IP downloading the file and from which port it was downloaded from.

```
sqlite> select external_ip,local_port,filepath from
conntrack,filetrack,files where conntrack.id = filetrack.cid AND
filetrack.fid = files.id AND filepath NOT NULL;
192.168.122.1|443|/root/dump/b_675318a2779027329cd8ac494885c41f.png
192.168.122.1|443|/root/dump/b_468934d7f5ad6d9c7774a838404dc985.png
192.168.122.1|443|/root/dump/b_06bffe74611bdd2b9ff872ddbce570ad.png
192.168.122.1|443|/root/dump/b_ff0a81287436e721541243946676c9d6.png
192.168.122.1|443|/root/dump/f_513e5dc74b2157cf435831158eca858e.jpg
192.168.122.1|443|/root/dump/f_82e71903da34a2f7b95af567e3041333.jpg
192.168.122.1|443|/root/dump/f_ccf11ad50a73c34da3a91ac7a03f82b9.jpg
192.168.122.1|443|/root/dump/f_51e2663a88befc0126b12ffddb896690.jpg
192.168.122.1|443|/root/dump/f_06b0f3f803bf320ba90b0b010c3068b2.jpg
192.168.122.1|443|/root/dump/f_53aa521e7c37e20d26ee118564b0c11c.jpg
192.168.122.1|443|/root/dump/f_656c2abe53056907e813d74a11aea29f.jpg
192.168.122.1|443|/root/dump/i_bb94b0324877d2daf7ec893075cb0824.jpg
192.168.122.1|443|/root/dump/i_bd5260c95858fbd196baf2e68575ab77.jpg
192.168.122.1|443|/root/dump/i_2bcfb46076d1cfcaefc2ceaaa0008fae.jpg
```

The output above is just a sample of the collected data. The full result is provided in Appendix H on page 75. As the sample shows the external IP (192.168.122.1) is correct, as it is the IP of the physical machine that was used as the client in the test. The port 443 is also correct as it is the SSL-enabeled Apache port on the target virtual machine that served the workload.

```
ls -1 /root/dump/
b_675318a2779027329cd8ac494885c41f.png
b_06bffe74611bdd2b9ff872ddbce570ad.png
b_468934d7f5ad6d9c7774a838404dc985.png
b_ff0a81287436e721541243946676c9d6.png
f_513e5dc74b2157cf435831158eca858e.jpg
f_82e71903da34a2f7b95af567e3041333.jpg
f_ccf11ad50a73c34da3a91ac7a03f82b9.jpg
f_51e2663a88befc0126b12ffddb896690.jpg
f_06b0f3f803bf320ba90b0b010c3068b2.jpg
f_53aa521e7c37e20d26ee118564b0c11c.jpg
f_656c2abe53056907e813d74a11aea29f.jpg
i_bb94b0324877d2daf7ec893075cb0824.jpg
i_bd5260c95858fbd196baf2e68575ab77.jpg
i_2bcfb46076d1cfcaefc2ceaaa0008fae.jpg
```

The output above is just a sample of the collected data. The full result is provided in Appendix I on page 77. As the output shows the extracted files

on the file system of the machine the prototype ran at is the same files that are registered as extracted by the prototype.

4.4 Coherence between analysis execution time and memory size

Three memory analysis commands (*linux_netstat*, *linux_lsof* and *linux_find_file*) are used by the prototype to gather and record information regarding ongoing file transfers. This section aims at getting a better understanding of how the total memory size of the virtual machine affects the execution time of the analysis commands.

A histogram displaying the average execution time of the command grouped by memory size has been created for each of the three commands. The error bars displayed in the histograms displays the 99.5% confidence interval based on the t-distribution because both the underlying distribution of the collected data and the standard deviation is unknown.

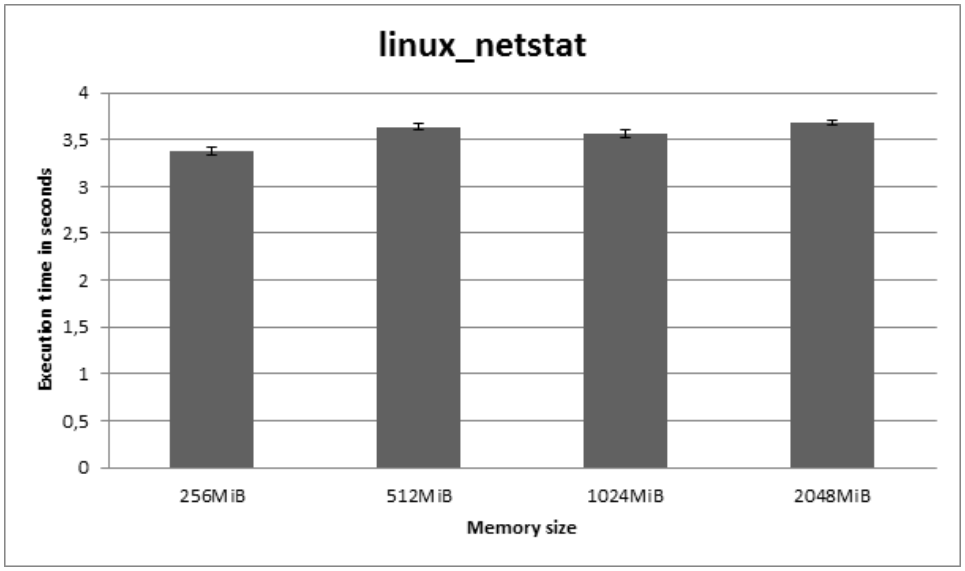

Figure 4.4: linux_netstat analysis execution time on four different memory configurations

The *linux_netstat* command is used to list current network connections on the target system and is the first step used to find ongoing file transfers.

The histogram displays an almost flat distribution with very narrow confidence intervals. The distribution suggests that the execution time does not increase significantly when increasing the memory size. Looking at the execution time of the 512 MiB and 2048 MiB configurations this can be seen quite clear. The 2048 MiB configuration consists of 400% more memory, yet the execution time is almost identical. However, the execution time of the lowest memory configuration seems to be a bit lower than the rest. The narrow confidence intervals suggests that the data that has been collected is very precise as when repeatedly taking samples the true average will be within the narrow confidence interval.

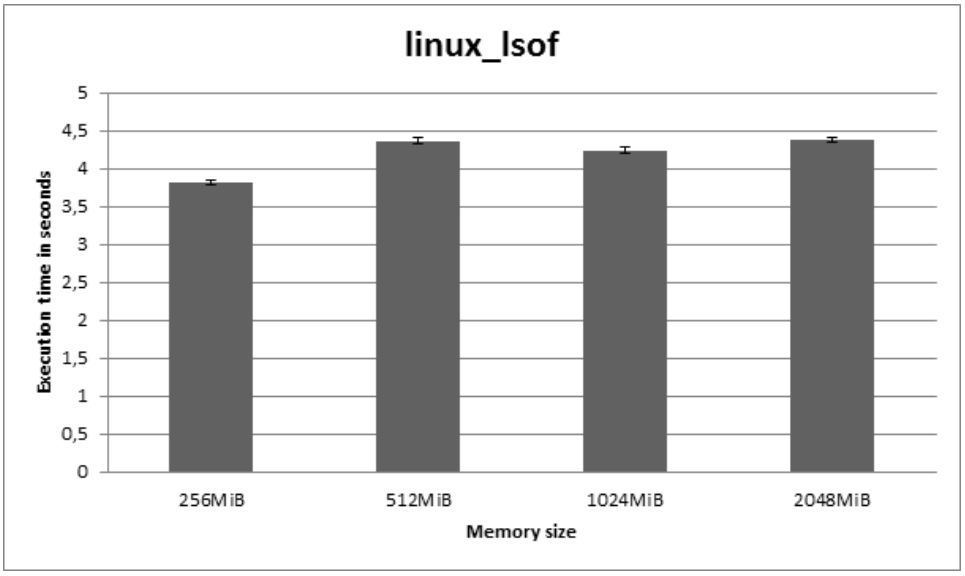

Figure 4.5: linux_lsof analysis execution time on four different memory configurations

The *linux_lsof* command is used to find open files associated with established network connections.

The histogram displays an almost flat distribution with very narrow confidence intervals. The result of this command is almost identical to the *linux_netstat* command and as such the analysis is almost the same. The execution time of the lowest memory configuration is however a bit lower compared to the rest of the configurations when running this command.

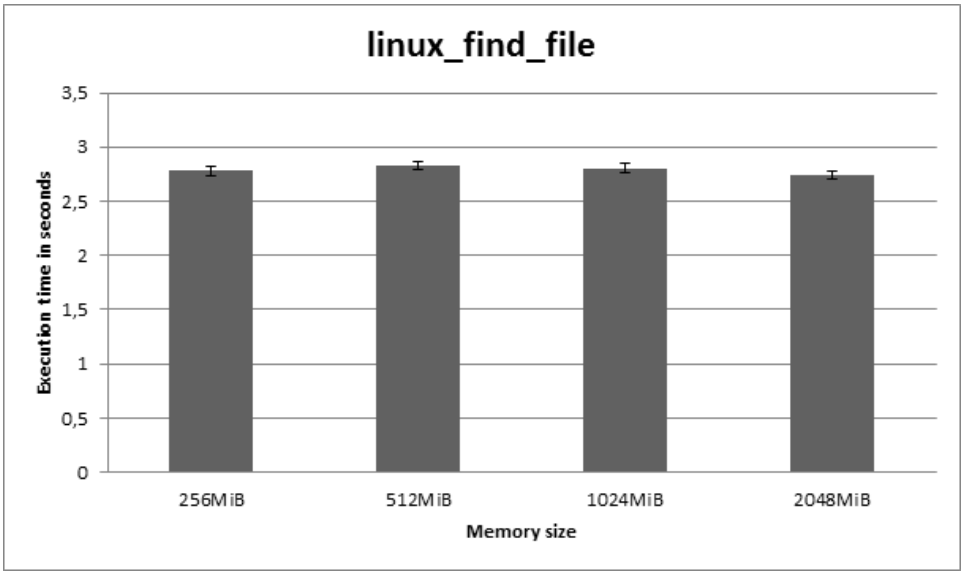

Figure 4.6: linux_find_file analysis execution time on four different memory configurations

The *linux_find_file* command is used to extract a file associated with a file transfer from the memory of the target.

The histogram displays an almost flat distribution with very narrow confidence intervals, the same as the previous commands. This suggests that the execution time is not penalized significantly by increasing the memory configuration.

4.5 Coherence between the number of extracted files and the network speed

The purpose of this test is to see how the network speed affects the number of files the prototype manages to extract. Since files will be downloaded faster when the network speed is increased they will consequently reside for a shorter time in memory. For that reason it is logical to assume that the number of extracted files will decrease when the network speed increase.

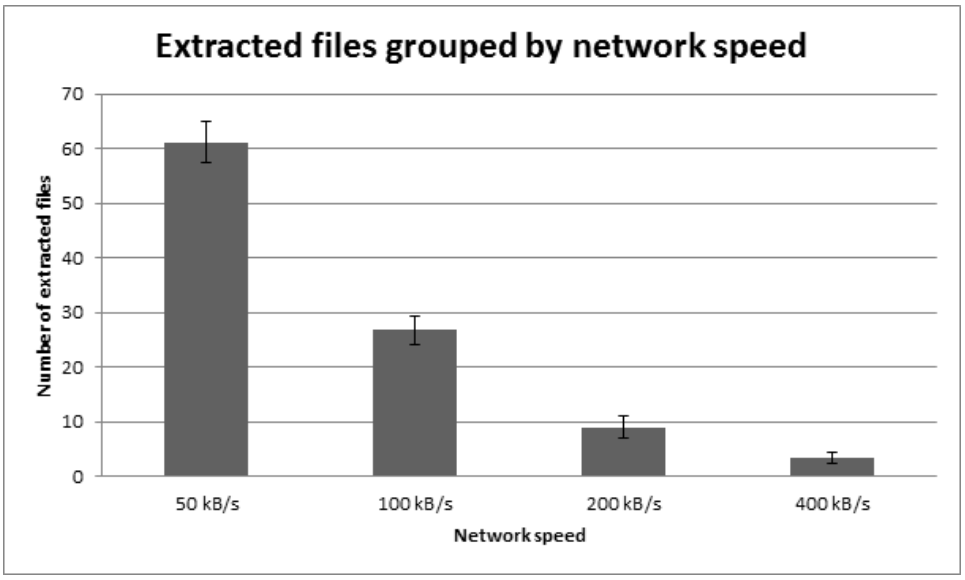

Figure 4.7: The number of extracted files on four different network speeds

The histogram above displays the average number of files extracted from memory grouped by the network speed. The error bars displays the 99.5% confidence interval based on the t-distribution because both the underlying distribution of the collected data and the standard deviation is unknown.

As the graph shows there is a big coherence between the network speed and the number of files the prototype manages to extract. Looking at the graph it almost looks like when the network speed double the amount of extracted files gets halved. Since the histogram can be hard to read a table with the exact calculated values is provided below:

Network Speed	50 kB/s	100 kB/s	200 kB/s	400 kB/s
Average file size in kB	61,13793	26,75862	9	3,344828
99.5% CI upper	64,86266	29,33665	11,1062	4,427646
99.5% CI lower	57,4132	24,18059	6,893797	2,262009

Table 4.1: The number of extracted files on four different network speeds

Using this table the multiplier of the decrease when doubling the

network speed can be found, as shown below:

- When the speed is increased from 50 kB/s to 100 kB/s the multiplier of the decrease of extracted files is 2.28.
- When the speed is increased from 100 kB/s to 200 kB/s the multiplier of the decrease of extracted files is 2.97.
- When the speed is increased from 200 kB/s to 400 kB/s the multiplier of the decrease of the extracted files is 2.69.

As shown the multiplier of the decrease in extracted files when doubling the network speed is within the range between 2.28 and 2.97, which can be seen as significant.

4.6 Coherence between average file size and network speed

The purpose of this test is to see if there is coherence between the average file size of the extracted files and the network speed the files are downloaded at. Since larger files take a longer time to transfer they will consequently remain longer in memory. Taking this into consideration it could be logical to assume that when you increase the network speed the average file size of the extracted files will also increase.

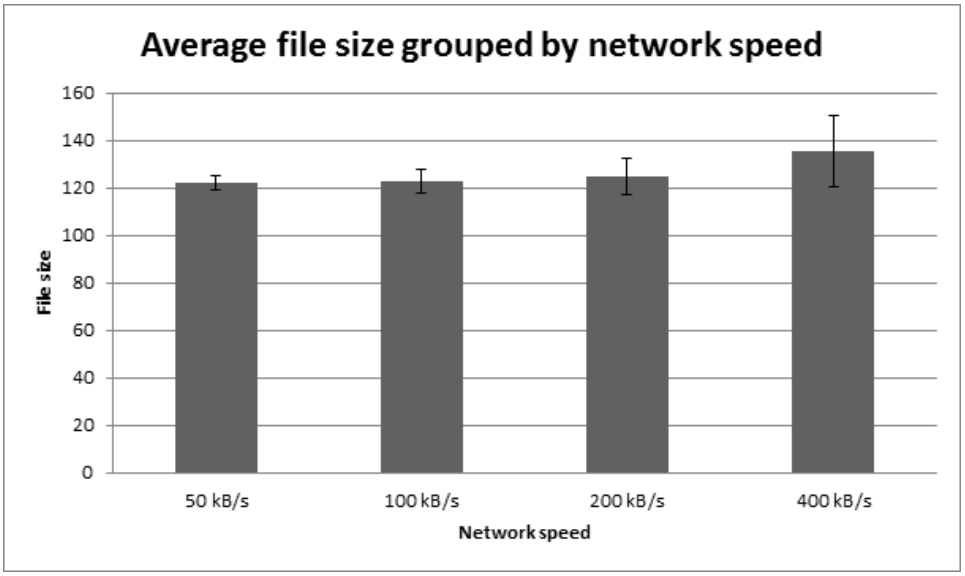

Figure 4.8: The average file size of extracted files on four different network speeds

The histogram above displays the average file size extracted from memory grouped by the network speed. The error bars displays the 99.5% confidence interval based on the t-distribution because both the underlying distribution of the collected data and the standard deviation is unknown.

The histogram shows that when the network speed increases the average file size of the extracted files also increase. This increase does not look that significant straight away, and when taking the error bars into account the distribution could actually be almost flat (the error bars on the faster speeds are quite broad). However, it is important to take the distribution of the workload's file sizes into account when evaluating these results. In section 4.2 it is shown that the distribution of file sizes is skewed towards the smaller file sizes, with the exception of some larger files. Taking this into account it is hard to argue that the increase is insignificant because it is hard to get a result with a high average file size when the distribution consists of mostly smaller files. Due to this it was chosen to provide the aggregated file size distribution of the tests done at the different network speeds.

4.6.1 Workload distribution

This is the workload file size distribution and the purpose is to use it as a reference when evaluating the aggregated file size distributions of the different network speeds. Since the network speed tests consisted of 30 samples each this distribution has been scaled to reflect that (multiplied by 30), which is why it is different from the workload distribution provided in section 4.2.

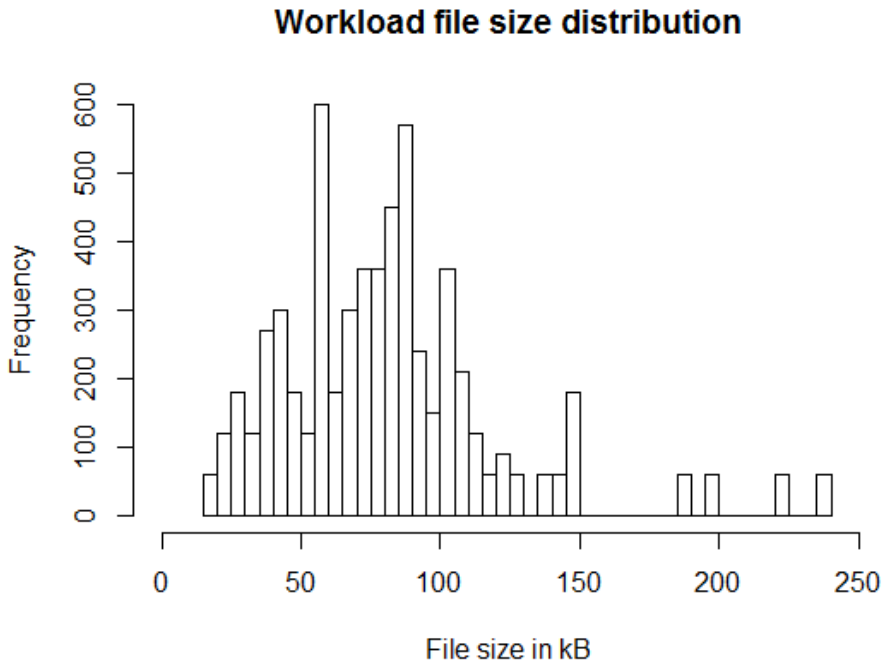

Figure 4.9: The workload frequency file size distribution

4.6.2 50 kB/s file size distribution

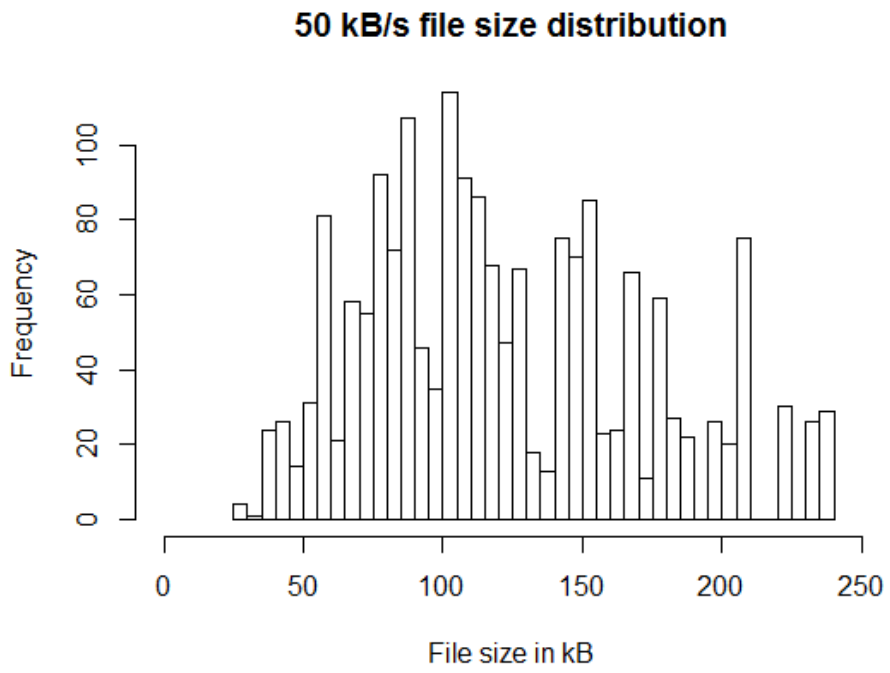

Figure 4.10: 50 kB/s frequency file size distribution

The histogram shows that the skew that was seen in the workload distribution is starting to become less evident. This suggests that the larger files are more likely to be extracted from memory as they are represented more than the original workload would suggest. This is logical as the larger files reside in memory longer than the smaller files. The smaller files are still in majority however.

4.6.3 100 kB/s file size distribution

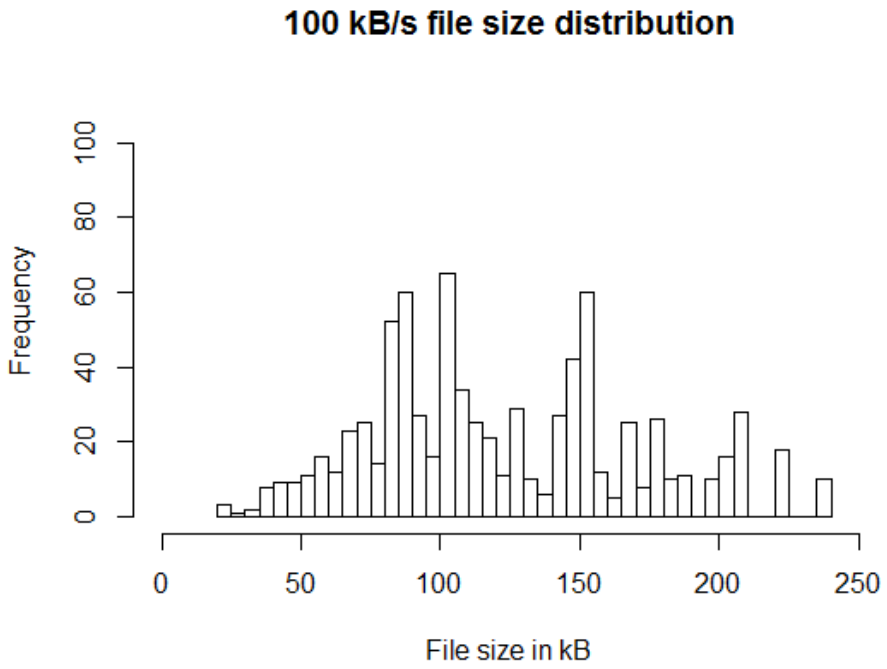

Figure 4.11: 100 kB/s frequency file size distribution

This histogram shows the same behavior as in the previous section and the original workload distribution is fading away even more. The smaller files are still in majority but it seems like the distribution is slowly starting to flat out.

4.6.4 200 kB/s file size distribution

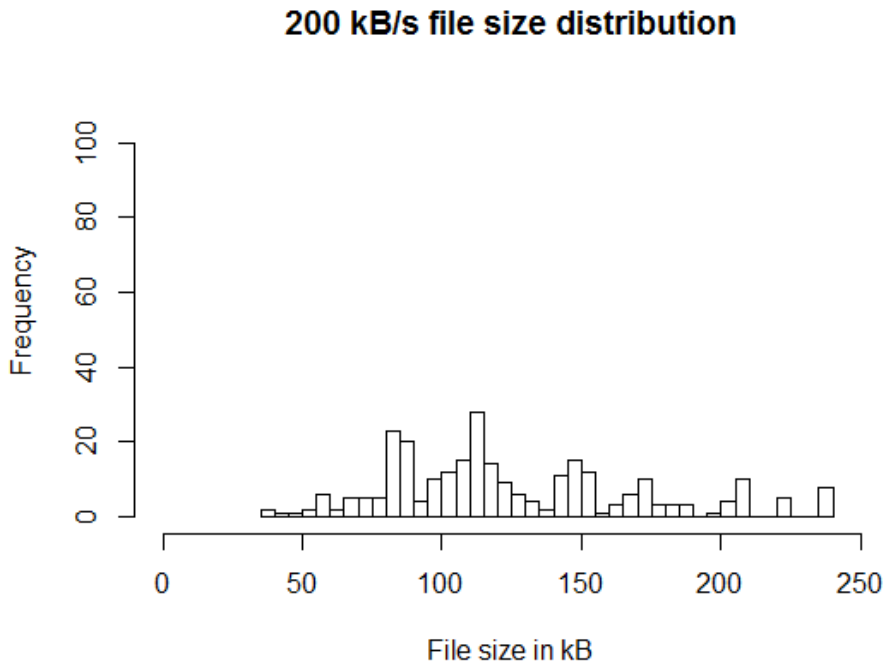

Figure 4.12: 200 kB/s frequency file size distribution

The distribution is flattening even more out but the smaller files are still the majority.

4.6.5 400 kB/s file size distribution

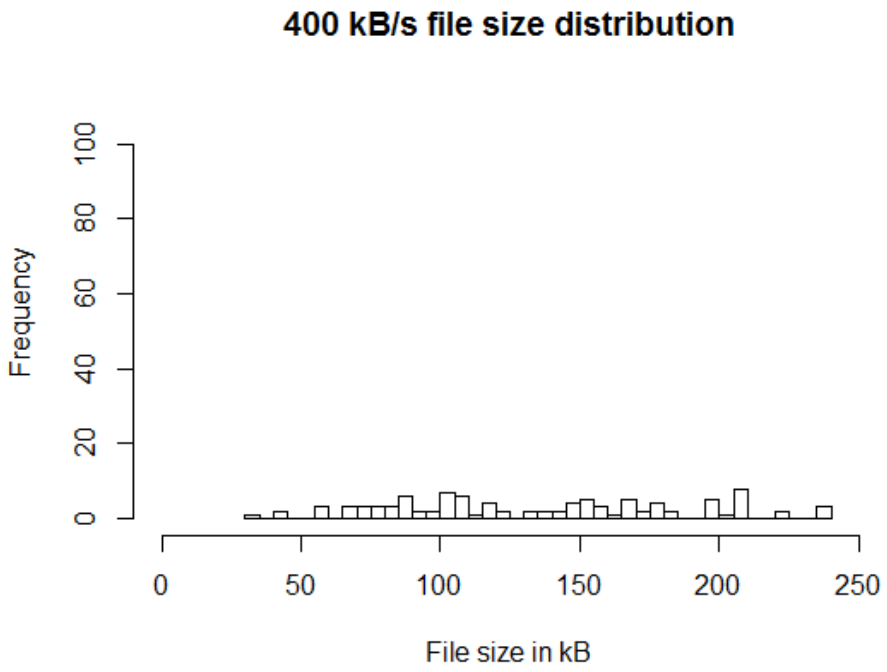

Figure 4.13: 400 kB/s frequency file size distribution

In this histogram the distribution is almost flat and it is hard to tell whether the smaller or larger files are in majority. By comparing this distribution to the original workload distribution it is easy to see that the average file size increases significantly when the network speed is increased. In the original workload distribution it was easy to see that the smaller files were in majority, which it no longer is.

Chapter 5

Discussion

The purpose of this thesis was to show that information that leaks between virtual machines and the VMM can be problematic. This is especially problematic now that cloud computing has gained momentum and the infrastructure the virtual machines are running on are administrated by a third party. This gives the customer little to no knowledge about who actually controls and possibly examines the resources they use. The focus has been on file confidentiality and the problem statement was narrowed down to see if it was possible to extract the files that are part of a file transfer on a virtual machine from its memory. A prototype was developed and a workload consisting of 200 images was created in order to test the prototype. This was done by having a client request the workload from a virtual machine serving it. Additionally the Red Hat virtualization security guideline was consulted to find best practices for file security in virtual environments. The guideline suggested that both the hard disks and the network traffic should be encrypted and as such the virtual machines serving the workload were set-up accordingly.

The workload was analyzed to find out how the file sizes were distributed and it was shown that the file size distribution was skewed to the left with the majority of the files being small (in the range between 20 and 125 kB), with some larger files (in the range between 200 and 250 kB). Since all further tests were done using the workload it is important to note that the results are only valid for this workload and that by running a different workload the results could differ.

The prototype was tested by having a client request the workload from a virtual machine over HTTPS, while the prototype was trying to extract the files from the memory of the virtual machine. The result showed that the prototype managed to extract some of the files. To confirm that the extracted files were the same as the files that were served an MD5 hash was calculated for all the extracted files and the files on the virtual machine. The MD5 hashes were then compared and it was shown that the files were in fact the same. This showed that it is possible to extract files that are part of a file transfer from the memory of a virtual machine, even when full disk

encryption and network SSL encryption is implemented on the virtual machine.

The reason for why the prototype is unable to extract all the files is that some time is spent analyzing the memory and updating the database with the findings. This leaves two time windows open where files can be transferred unnoticed by the prototype. There are several properties that might affect the length of the time windows, some of which were measured and is discussed below.

The memory configuration of the virtual machine could possibly affect the time the prototype spends analyzing the memory. This is because it could take a longer time to analyze a larger memory configuration than a smaller one. By measuring the execution time of the three memory analysis commands used in the prototype on different memory configurations, with the largest being 800% larger than the smallest, it was shown that the memory configuration did not affect the execution time significantly.

The network speed the workload is downloaded at could possibly reduce the number of extracted files as well. This is due to the fact that the files will reside in memory for a shorter time when the network speed is increased (because the files will be downloaded faster). It was shown by measurements that when doubling the network speed the number of extracted files will decrease by a multiplier of between 2.28 and 2.97. This shows that doubling the network speed have a significant effect on the prototype, as the decrease in files it will be able to extract more than halves. It is however important to note that the workload consisted of a lot of small files and that the effect could have been less significant if the workload consisted of larger files.

It was also tested if the average file size of the extracted files would increase when the network speed was increased. The reasoning behind that assumption is that larger files will reside longer in memory and as such will be more likely to be found than small files when the network speed is increased. The measurements showed that the average file size did increase significantly. Due to the fact that the workload consisted of mostly smaller images it could be that the average file size could increase even more significantly if the workload consisted of larger files.

Chapter 6

Conclusion

In order to conclude the problem statement that was set for this thesis is provided, the findings are summarized and further work is discussed.

The problem statement of the thesis was the following:

How can data that is secured with encryption on a virtual machine be captured by the VMM?

Throughout this thesis it has been shown, by creating a prototype that combines several already existing tools, that it is possible to extract the file(s) that are part of an ongoing file transfer on a virtual machine from its memory. It was also shown that this can be done even when fully encrypting the disk and network traffic of the virtual machine and that the process is transparent to the owner of the machine. The number of files that can be extracted are however limited by the memory analysis that has to happen real time. This opens a time window between each time the memory analysis is ran, which means that some files will escape disclosure because they are transmitted within the open time window. However, it has further been shown, through measurements, that there are at least two factors that significantly affect the number of files that can be extracted. The first factor is the file size, which is due to the fact that larger files will reside for a longer time in memory and as such the chance of disclosure for those files are higher. The second factor is the network speed at which the files are downloaded, which also affects how long the files will reside in memory (the faster you download the shorter the file will reside in memory).

Having concluded there is a lot of interesting work that can be done in the future to further explore the possibilities of file transfer extractions from memory. The prototype created for this thesis uses a combination of Volatility analysis commands to analyze the memory and in turn extract the files. This is not an efficient approach, but is efficient enough to prove that file extraction is possible and to measure some of the factors that affect such an extraction. By creating the analysis command from scratch, with the sole purpose of file transfer extraction, the prototype can probably be improved

a lot. Additionally, only a single workload using a single connection with relatively small files was used to test and measure the prototype. By doing the same measurements using other workloads with larger files and several connections one might be able to find other factors that affect the prototype.

While the focus of this thesis has been to enable extraction of files from ongoing file transfers from memory, even in the event that the administrator has set-up the machine according to current guidelines, it would be interesting to research if there are ways for administrators to limit the efficiency of the prototype. Since the files are contained in caches in memory it would be interesting to start the research by looking at kernel parameters that can adjust the file caches in memory, to see if that can limit the ability to extract the files.

Appendix A

wrapper.pl

```
1  #!/usr/bin/perl
2  use strict;
3  use threads;
4
5  my $path = "/path/to/scripts";
6  my $conntrack = $path . "conntrack.pl";
7  my $filetrack = $path . "filetrack.pl";
8  my $filedumper = $path . "filedumper.pl";
9
10 $thr1 = threads->create('run', $conntrack);
11 $thr2 = threads->create('run', $filetrack);
12 $thr3 = threads->create('run', $filedumper);
13 $thr1->join();
14 $thr2->join();
15 $thr3->join();
16 sub run{
17     my $cmd = shift();
18     system($cmd);
19 }
```

Appendix B

conntrack.pl

```
1  #!/usr/bin/perl
2  use strict;
3  use DBI;
4  use Digest::MD5 qw(md5 md5_hex md5_base64);
5  use Getopt::Std;
6
7  # Debug off = 0, Debug on = 1
8  my $debug = 1;
9
10 # Volatility configuration
11 my %volatility = (
12     bin => '/root/volatility-2.3.1/vol.py',
13     command => 'linux_netstat',
14     file => 'vmi://ubuntu1024',
15     profile => 'Linuxubuntu-13_10-serverx64'
16 );
17
18 # DB configuration
19 my %db = (
20         driver => 'SQLite',
21         database => 'db.db',
22         username => '',
23         password => ''
24     );
25
26 # DB tables configuration
27 my %dbTables = (
28             conntrack => 'conntrack',
29             files => 'files',
30             filetrack => 'filetrack'
31 );
32
33 # Trying to connect to the DB
34 my $dsn = "dbi:" . $db{driver} . ":dbname=" . $db{database};
35 my $dbh = DBI->connect($dsn, $db{username}, $db{password}) or die
   $DBI::errstr;
36 print "DEBUG: Connected to database \n" if($debug);
37
38 # Prepairing SQL statements
39 my $openConnectionsQuery = $dbh->prepare("SELECT hash FROM" . " "
   . $dbTables{conntrack} . " " . "WHERE closed IS NULL;");
```

```
40 my $openConnectionAdd = $dbh->prepare("INSERT INTO" . " " .
   $dbTables{conntrack} .
   "(protocol,local_ip,local_port,external_ip,external_port,pid,
41 application,created,updated,hash) VALUES (?,?,?,?,?,?,?,?,?,?)");
42 my $openConnectionUpdate = $dbh->prepare("UPDATE" . " " .
   $dbTables{conntrack} . " " . "SET updated = ? WHERE hash = ?");
43 my $openConnectionClose = $dbh->prepare("UPDATE" . " " .
   $dbTables{conntrack} . " " . "SET closed = ? WHERE hash = ?");
44
45 # Fetching already open connections.
46 # These might exist if the program unexpectedly shut down at some
   point.
47 my %openConnections;
48 $openConnectionsQuery->execute();
49 foreach(@{$openConnectionsQuery->fetchall_arrayref()}){
50     $openConnections{@{$_}[0]} = undef;
51 }
52 print "DEBUG: Found " . scalar(keys %openConnections) . " open
   connections in the DB\n" if($debug);
53
54 my $counter = 0;
55
56 while(){
57 $counter++;
58 print "Run # " . $counter . "\n" if($debug);
59 # Using volatility to find currently established/open connections
60 my $command = $volatility{bin} . " " . $volatility{command} . "
   -l " . $volatility{file} . " --profile=" . $volatility{profile} .
   " 2>/dev/null";
61 my $execStartTime = time();
62 my @result = grep(/ESTABLISHED/,qx($command));
63 print "DEBUG: Found " . scalar(@result) . " currently open
   connections\n" if($debug);
64
65 # Going through the currently open connections.
66 # If the connection already exists in the DB, the update field is
   updated with the current timestamp.
67 # If the connection doesnt exist in the DB, the connection entry
   will be added.
68 # If the connection is no longer present the closed timestamp is
   updated in the DB.
69 foreach my $connection(@result){
70     my $hash = md5_hex($connection);
71     if(exists($openConnections{$hash})){
72         $openConnectionUpdate->execute(time(),$hash);
73         print "\t Connection with hash $hash already exists in
           the DB, updating the update timestamp\n" if($debug);
74     }else{
75         my ($protocol,$local,$external,$state,$pidpro) =
           split(/\s+/,$connection);
76         my ($localIp,$localPort) = split(/:/,$local);
77         my ($externalIp,$externalPort) = split(/:/,$external);
78         my ($application,$pid) = split(/\//,$pidpro);
79         my $createdTime = time();
80         $openConnectionAdd->execute($protocol,$localIp,$localPort,
81         $externalIp,$externalPort,$pid,
82         $application,$createdTime,$createdTime,$hash);
83         print "\t Connection with hash $hash doesnt exists in the
           DB, adding entry\n";
```

```
84      }
85      $openConnections{$hash} = $execStartTime;
86  }
87  # Closing connections that are no longer present
88  foreach my $hash (keys %openConnections){
89      if($openConnections{$hash} ne $execStartTime){
90          $openConnectionClose->execute(time(),$hash);
91          delete($openConnections{$hash});
92          print "\t Connection with hash $hash is no longer
            present, updating the closed timestamp\n" if($debug);
93      }
94  }
95
96  }
97  $dbh->disconnect;
```

Appendix C

filetrack.pl

```
1  #!/usr/bin/perl
2  use strict;
3  use DBI;
4  use Getopt::Std;
5
6  # Debug off = 0, Debug on = 1
7  my $debug = 1;
8
9  # Volatility configuration
10 my %volatility = (
11      bin => '/root/volatility-2.3.1/vol.py',
12      command => 'linux_lsof',
13      file => 'vmi://ubuntu1024',
14      profile => 'Linuxubuntu-13_10-serverx64'
15 );
16
17 # DB configuration
18 my %db = (
19      driver => 'SQLite',
20      database => 'db.db',
21      username => '',
22      password => ''
23 );
24
25 # DB tables configuration
26 my %dbTables = (
27          conntrack => 'conntrack',
28          files => 'files',
29          filetrack => 'filetrack'
30 );
31
32 # File excludes
33 my %fileExcludes = (
34      apache2 => ["\/",
35          "\\[\\]",
36          "\/dev\/null",
37          "\/var\/log\/apache2\/other_vhosts_access.log",
38          "\/var\/log\/apache2\/ssl_access.log",
39      "\/var\/log\/apache2\/ssl_error.log",
40          "\/var\/log\/apache2\/access.log",
41          "\/var\/log\/apache2\/error.log",
42          "\/run\/apache2\/ssl_mutex"]
```

```
43 );
44
45 # Mappings
46 my %pidCid;
47 my %pidApp;
48
49 # Trying to connect to the DB.
50 my $dsn = "dbi:" . $db{driver} . ":dbname=" . $db{database};
51 my $dbh = DBI->connect($dsn, $db{username}, $db{password}) or die
   $DBI::errstr;
52 print "DEBUG: Connected to database (" . $db{database} . ") using
   the " . $db{driver} . " driver \n" if($debug);
53
54 # Prepairing SQL statements
55 my $openConnectionsQuery = $dbh->prepare("SELECT
   pid,id,application FROM" . " " . $dbTables{conntrack} . " " .
   "WHERE closed IS NULL;");
56 my $cidAddedFilesQuery = $dbh->prepare("SELECT " .
   $dbTables{files} . ".filename FROM " . $dbTables{files} . "," .
   $dbTables{filetrack} . " WHERE " . $dbTables{files} . ".id = " .
   $dbTables{filetrack} . ".fid AND " . $dbTables{filetrack} . ".cid
   = ?");
57 my $fileAdd = $dbh->prepare("INSERT INTO" . " " .
   $dbTables{files} . " " . "(filename,created) VALUES (?,?)");
58 my $filetrackAdd = $dbh->prepare("INSERT INTO" . " " .
   $dbTables{filetrack} . " " . "(fid,cid) VALUES (?,?)");
59
60 my $counter = 0;
61
62 while(){
63 # Mappings
64 my %pidCid;
65 my %pidApp;
66
67 $counter++;
68 print "Run # " . $counter . "\n" if ($debug);
69 # Going through all open connections and doing the following:
70 # Mapping the connection id to the process id, so that we know
   which connection(s) that are associated with a process.
71 # Mapping the process id to the application name (eg. apache), so
   that we know the application name of the process.
72 $openConnectionsQuery->execute();
73 foreach my $ref (@{$openConnectionsQuery->fetchall_arrayref()}){
74     my ($pid,$cid,$app) = @{$ref};
75     push(@{$pidCid{$pid}},$cid);
76     $pidApp{$pid} = $app;
77 }
78
79 # Checking that there are processes with open connections before
   doing any further processing.
80 if(scalar(keys(%pidCid)) > 0){
81     # Putting together the volatility command to list all open
       files on the system.
82     my $command = $volatility{bin} . " " . $volatility{command} .
       " -l " . $volatility{file} . " --profile=" .
       $volatility{profile} . " 2>/dev/null";
83     # Running the command and filtering the result to only
       contain files opened by processes that has open connections.
84     my $pids = "(" . join("|",keys(%pidCid)) . ")";
```

```
85      my @result = grep(/^\s+$pids/,qx($command));
86      print "DEBUG: Found " . scalar(keys(%pidCid)) . " process(es)
        with open connection(s)\n" if($debug);
87      print "DEBUG: The process(es) has " . scalar(@result) . "
        open file(s) in total before exclusion\n" if ($debug);
88      # Going through the processes to see if they have any open
        files.
89      # If they do and the files are not associated with
        connections in the DB we add them.
90      print "DEBUG: looping processes..\n" if($debug);
91      foreach my $pid (keys(%pidCid)){
92          my $app = $pidApp{$pid};
93          my @files = grep(/^\s+$pid/,@result);
94          # Excluding files if the exclude list is present.
95          if(defined($fileExcludes{$app})){
96              my $excludes = "(" . join("|",@{$fileExcludes{$app}})
                . ")";
97              @files = grep(!/$excludes$/,@files);
98          }
99          print "\t PID: $pid ($app) has " .
            scalar(@{$pidCid{$pid}}) . " open connections (cid: " .
            join(",", @{$pidCid{$pid}}) . ") and " . scalar(@files) .
            " open file(s) after exclusion\n" if($debug);
100         # Adding open files (if any) to a hash array.
101         next if($#files <1);
102         my %pidFiles;
103         foreach my $line(@files){
104             $line =~ /^.+\d+\s+(.+)\s+$/;
105             $pidFiles{$1} = undef;
106         }
107         # Going through the open connections to check which files
            that are already added to the DB (if any).
108         foreach my $cid (@{$pidCid{$pid}}){
109             $cidAddedFilesQuery->execute($cid);
110             # Putting the files retreieved from the DB into a
                hash array.
111             my %cidFiles = map { $_ => undef }
                @{$dbh->selectcol_arrayref($cidAddedFilesQuery)};
112             # Going through all files opened by the process and
                adding the connection id if the file is not present
                in the DB.
113             foreach my $pidFile(keys(%pidFiles)){
114                 push(@{$pidFiles{$pidFile}},$cid)
                    if(!exists($cidFiles{$pidFile}));
115             }
116         }
117         # Looping the process files that has to be added to the
            DB.
118         foreach my $file (keys(%pidFiles)){
119             next if(!defined($pidFiles{$file}));
120             $fileAdd->execute($file,time());
121             my $fid =
                $dbh->last_insert_id(undef,undef,"files",undef);
122             # Adding the file to the connection ID.
123             foreach my $cid(@{$pidFiles{$file}}){
124                 $filetrackAdd->execute($fid,$cid);
125             }
126             print "\t\t Aded file: $file to cid(s): " .
                join(",",@{$pidFiles{$file}}) . "\n" if($debug);
```

```
127             }
128         }
129 }
130
131 }
132 $dbh->disconnect;
```

Appendix D

filedumper.pl

```
1  #!/usr/bin/perl
2  use strict;
3  use DBI;
4  use Digest::MD5 qw(md5 md5_hex md5_base64);
5  use Getopt::Std;
6
7  # Debug off = 0, Debug on = 1
8  my $debug = 1;
9
10 # Volatility configuration
11 my %volatility = (
12      bin => '/root/volatility-2.3.1/vol.py',
13      command => 'linux_find_file',
14      file => 'vmi://ubuntu1024',
15      profile => 'Linuxubuntu-13_10-serverx64'
16 );
17
18 # DB configuration
19 my %db = (
20      driver => 'SQLite',
21      database => 'db.db',
22      username => '',
23      password => ''
24 );
25
26 # DB tables configuration
27 my %dbTables = (
28          conntrack => 'conntrack',
29          files => 'files',
30          filetrack => 'filetrack'
31 );
32
33 my $dumpDir = "/root/dump";
34
35 # Trying to connect to the DB.
36 my $dsn = "dbi:" . $db{driver} . ":dbname=" . $db{database};
37 my $dbh = DBI->connect($dsn, $db{username}, $db{password}) or die
   $DBI::errstr;
38 print "DEBUG: Connected to database (" . $db{database} . ") using
   the " . $db{driver} . " driver \n" if($debug);
39
40 # Prepairing SQL statements
```

```
41 my $openFilesQuery = $dbh->prepare("SELECT
   files.id,files.filename FROM files,filetrack,conntrack WHERE
   files.id=filetrack.fid AND filetrack.cid=conntrack.id AND
   conntrack.closed IS NULL AND files.fileadded IS NULL;");
42 my $fileAdd = $dbh->prepare("UPDATE" . " " . $dbTables{files} . "
   " . "SET fileadded = ?, filepath = ? WHERE id = ?");
43
44 # Volatility commands
45 my $volatilityBaseCommand = $volatility{bin} . " " .
   $volatility{command} . " -l " . $volatility{file} . " --profile="
   . $volatility{profile};
46
47 # Going through files where the connection is still open and the
   file not yet dumped.
48 my $counter = 0;
49
50 while(){
51     $counter++;
52     print "Run #" . $counter . "\n" if($debug);
53     my %openFiles;
54     $openFilesQuery->execute();
55     foreach my $ref(@{$openFilesQuery->fetchall_arrayref()}){
56         my ($fid,$filename) = @{$ref};
57         push(@{$openFiles{$filename}},$fid);
58     }
59     foreach my $file(keys(%openFiles)){
60         print "Found file $file, trying to dump it:\n" if($debug);
61         my $volatilityFindInodeCommand = $volatilityBaseCommand .
           " -F " . $file . " 2>/dev/null";
62         # Finding the inode of the current file
63         my @result = qx($volatilityFindInodeCommand);
64         if($#result>0){
65             my $inode = (split(/\s+/,$result[$#result]))[2];
66             print "\t Inode is $inode\n" if($debug);
67             my @fileName = split(/\//,$file);
68             my $path = $dumpDir . "/" . $counter . "_" .
               $fileName[$#fileName];
69             my $volatilityDumpFileCommand =
               $volatilityBaseCommand . " -i " . $inode . " -O " .
               $path . " 2>/dev/null";
70             # Updating the DB if the file gets successfully
               dumped.
71             if(system($volatilityDumpFileCommand) == 0){
72                 foreach my $fid(@{$openFiles{$file}}){
73                     $fileAdd->execute(time(),$path,$fid);
74                     print "\t Dumped to $path\n" if($debug);
75                 }
76             }else{
77                 print "\t Couldnt be dumped\n" if($debug);
78             }
79         }else{
80             print "\t Could not find inode\n" if ($debug);
81         }
82     }
83 }
84 $dbh->disconnect;
```

Appendix E

Database create script

```
CREATE TABLE conntrack(
id INTEGER PRIMARY KEY AUTOINCREMENT,
protocol TEXT NOT NULL,
local_ip TEXT NOT NULL,
local_port INT NOT NULL,
external_ip TEXT NOT NULL,
external_port INT NOT NULL,
pid INT NOT NULL,
application TEXT NOT NULL,
created INT NOT NULL,
updated INT,
closed INT,
hash TEXT NOT NULL);
CREATE TABLE files(
id INTEGER PRIMARY KEY AUTOINCREMENT,
filename TEXT NOT NULL,
created INT NOT NULL,
fileadded INT,
filepath TEXT);
CREATE TABLE filetrack(
id INTEGER PRIMARY KEY AUTOINCREMENT,
fid INTEGER NOT NULL,
cid INTEGER NOT NULL,
FOREIGN KEY(fid) REFERENCES files(id),
FOREIGN KEY(cid) REFERENCES conntrack(id));
```

Appendix F

Httperf workload

```
/blogspot_9885699381d6d5e9bbaf41aa2da50e84.png
/blogspot_d771302591dbf18db86a140e7f078d79.png
/blogspot_003a03b988cd4ebb695ef72ba4009139.png
/blogspot_75bfe8b9704630ec6b8e6aaccc10ab4a.png
/blogspot_675318a2779027329cd8ac494885c41f.png
/blogspot_fe0eda22fd9aa069f4fa0a4901c3d91d.png
/blogspot_6e23784acc955d377c45af7e1b8be611.png
/blogspot_0bd484fdbe637f08ece60facbd831c1d.png
/blogspot_8683af34e29e622f425111deb04c9d85.png
/blogspot_99e5e1c2d18bc1ad0ca609826d824617.png
/blogspot_15e5ead898c23375754d674aa05ef589.png
/blogspot_9c0c88ad882ad2410e9bbde83b2de5b9.png
/blogspot_e1be0cab3c4bc044c31ac0219c309fa4.png
/blogspot_f98f3202f71dca3ffdb6c694f9053e06.png
/blogspot_ca0a69a05f6a295538a666980b539809.png
/blogspot_2b927f2d2732678c20332d9b320e5718.png
/blogspot_9342e3e875abb0fcb7f4fe9eb8f29ff4.png
/blogspot_ad0e815a03727bd6b67a527fea4b5725.png
/blogspot_468934d7f5ad6d9c7774a838404dc985.png
/blogspot_5d062a04ec078a2c6b27474c1551a90b.png
/blogspot_f9e31735359fe15ff8cd140b4c27d3a8.png
/blogspot_c9bfa5c24f998e0f278bb474a79e8ba5.png
/blogspot_a0cb46593e506eb97c7700a9c3dd166c.png
/blogspot_06bffe74611bdd2b9ff872ddbce570ad.png
/blogspot_d84f42774a5fba91273925c03b08a3f6.png
/blogspot_cfb70669279065a186c1355c55271fc9.png
/blogspot_937550eaa5a8ca428bd4fd9b832907aa.png
/blogspot_b19dc918dbe0feb33eea7e262f34c6ee.png
/blogspot_c7df9abf834b7fd8951df741a537f02f.png
/blogspot_1f748c2f3e8562118f07337753cadd37.png
/blogspot_6fc962d04b23a18420dfdc22cd7f6647.png
/blogspot_37fec66aacf87acf0c8cb6127e654e26.png
/blogspot_883aa4a506ba8fd6223988002978d2d2.png
/blogspot_a693fabd9283b5836f69437f35640ebb.png
/blogspot_43fa3cb5f1ecfb9e7975edc20dab823f.png
/blogspot_f3db0ab6f59fa0222cdb7dc7ff6e1780.png
/blogspot_ccd83fe09d9e4a992d93f0111308d0c3.png
/blogspot_d8be334b474a951e92ae5d2890189ed1.png
/blogspot_9736826cad0432ab51868c4b76ce2488.png
/blogspot_b609ff286420f021fadcecc7728c1a22.png
/blogspot_6e87dfa2d532af8f279b98473b3abd0a.png
/blogspot_33721dfb64ba4ae42f0dc4bc620e5c81.png
```

```
/blogspot_d0cd3da4bda3ad377d6ff48150b84184.png
/blogspot_ec92dafac3fa3d26712b56df495069fa.png
/blogspot_bcd1be9e6d1862aad5fb0b6fe09fcccc.png
/blogspot_79109942ff91936f0866cb7f72019014.png
/blogspot_a3c81f9ad62216f3c9eecbfe0cf247e8.png
/blogspot_ecd6d43b0ad327910282667120054772.png
/blogspot_db2fecc36beba2b498945b2ef62a3c72.png
/blogspot_ff0a81287436e721541243946676c9d6.png
/facebook_c6ee0a44956afb0770136f4fe01b9c5a.jpg
/facebook_513e5dc74b2157cf435831158eca858e.jpg
/facebook_d3860910e39fefca19f50b2962f79c04.jpg
/facebook_bed37a7b602ab0994417748e83637214.jpg
/facebook_82e71903da34a2f7b95af567e3041333.jpg
/facebook_f3b5780d89d14394ce91a4c09ee896e1.jpg
/facebook_954a6c713a2bc9a2fadda7c54ff44278.jpg
/facebook_459d0a8889340be05f7e784b3ee55504.jpg
/facebook_1b49d78060286d3c2d46e5b8c9283ebf.jpg
/facebook_4fee94a5ced35ddcc02ea7ceae96d8e1.jpg
/facebook_35a38edce23857b891ec60d36bbe2a4f.jpg
/facebook_b80fd2516536d444d6d7894acf9d4eed.jpg
/facebook_628efb5c558aa47eda8a8929da4779a1.jpg
/facebook_19c805b6ce6c71d67ccd25ecdde90129.jpg
/facebook_b9a3509722563e853897fc18436f219e.jpg
/facebook_8304e7418f402241d01709d93753b2e2.jpg
/facebook_38dc4e902d477e0b98a4781812cff4d3.jpg
/facebook_ccf11ad50a73c34da3a91ac7a03f82b9.jpg
/facebook_a50c7fcfa98f217f2185c216b7a29b22.jpg
/facebook_e181e1902b76c374835127a6506f04eb.jpg
/facebook_51e2663a88befc0126b12ffddb896690.jpg
/facebook_ced2d2a25d2b73277eb005b173d376b4.jpg
/facebook_1b96776ab4452f6b24aa37306bd390eb.jpg
/facebook_22b12198f3a1b6cd44b636822bbef2a4.jpg
/facebook_ce94224344d86d1bd302804b5ea52017.jpg
/facebook_08e584806e4e5ecd6057aafaeee85182.jpg
/facebook_956a42f4dd2d72dbf4a6dcae82dd6a2f.jpg
/facebook_df969dc994c06a494a63bea965f2b25b.jpg
/facebook_1bb99895d83a27e62b76ec3603d7670a.jpg
/facebook_1b539abaa312ec4ac2c6c4a2716e1013.jpg
/facebook_7442c6fde17b08c524198b6714e19576.jpg
/facebook_df159e306967df2d09cdc8642a572f17.jpg
/facebook_eeb88c04968bf76cf18310745696d6c7.jpg
/facebook_4c5c75051f6b45991c512dfda57f8d14.jpg
/facebook_06b0f3f803bf320ba90b0b010c3068b2.jpg
/facebook_6df8fedd476f7f87bf07c9aa2e12105f.jpg
/facebook_9efe99c9fc2b9e6b3934676c7570c6aa.jpg
/facebook_a5041449947c23f7d5e44389361b8239.jpg
/facebook_ca220e5dab1a20190ac5d0dcf73ce45b.jpg
/facebook_7c266e30ec2dd0c2b92ebddd41ba0587.jpg
/facebook_ac30d297c360d3ba1ea5e427a256f0a2.jpg
/facebook_b71bc076de50bb8a77a05636dfd8105d.jpg
/facebook_53aa521e7c37e20d26ee118564b0c11c.jpg
/facebook_40db1e1b083ba012b36216c9f8fe60b9.jpg
/facebook_656c2abe53056907e813d74a11aea29f.jpg
/facebook_9322b2f19cca3f7bf7a31176d7c4947c.jpg
/facebook_5bd0039865f62f10ace1e20f92e89049.jpg
/facebook_beb1a1bfd2b591fcda88275683a716ba.jpg
/facebook_4c0cf3f7293d99dfb741f6fd27a9a245.jpg
/facebook_6f3a0fe3af9af7393cc6eee498c58070.jpg
/instagram_a88a935aca5b7565757886635ce12489.jpg
```

```
/instagram_685dfc68c8f516344b02619e5f4c68d9.jpg
/instagram_2bc17f12995eef3a256800f3c00fe378.jpg
/instagram_7e4557d15a25bc34eb717316d166aa57.jpg
/instagram_bb94b0324877d2daf7ec893075cb0824.jpg
/instagram_1d666f7cc9f05669bfb1903179f861e5.jpg
/instagram_53408c6bdf65ac322f62b292ff093907.jpg
/instagram_bd5260c95858fbd196baf2e68575ab77.jpg
/instagram_ee8af722feb02858ea44fd015b008213.jpg
/instagram_6b80be27f0ef59d2fc00c1ab23866a2f.jpg
/instagram_bb3d311560ea247040eafe953fb1a2fb.jpg
/instagram_cef222c9af30fdd014a577e6fb1edab2.jpg
/instagram_2bcfb46076d1cfcaefc2ceaaa0008fae.jpg
/instagram_e73332c15f03527a64e0972d9286a405.jpg
/instagram_6a34a0a525ec9e57a2feea3eac6eb4cd.jpg
/instagram_a07a141cbbd08579bd7a265267f2108a.jpg
/instagram_a6c5403844eb25ce118ea0367d66ba81.jpg
/instagram_cab78f19dae041cd51ffa00376bad0b8.jpg
/instagram_b672b724b3417d93bfcbd6d5bf425733.jpg
/instagram_e900c2e5344393a393bbede0ee14e507.jpg
/instagram_779f652d652005665da15f37bb761276.jpg
/instagram_92b653525b8b00a08b5f5dfd9a687b5e.jpg
/instagram_b90355d24b7354de6d4b6de1513f3abf.jpg
/instagram_552de08bc61f111b0f73d67f3c3e0299.jpg
/instagram_0c5e909fc545385b2e01ec8665e94a5f.jpg
/instagram_518a5c9fcf876beac56118f0accfe6f5.jpg
/instagram_fcabb4451922d28b0dbd8211db865d8f.jpg
/instagram_94737932872d5f72d0ac53bf41b867f5.jpg
/instagram_2bcf1f091096412b9c048a2e8cd4e933.jpg
/instagram_3c57861b831cf75b434b577dc7a36741.jpg
/instagram_2b143a6012425baf3cd3e62426c72b7f.jpg
/instagram_a4c033132eb120f377d44a6f5c8879a2.jpg
/instagram_d91ea58dd5ddc0283fc17f15042b6d45.jpg
/instagram_a8dd9c82e2bb4394f837199a9737afb5.jpg
/instagram_ce0a484421a0d0787c1065207c4361fb.jpg
/instagram_107a12d31c387d2d9ea74a997330b261.jpg
/instagram_d5c50ba013d77f95835f840535508e26.jpg
/instagram_5380a777f17f38a87d3f1b495c4add1b.jpg
/instagram_75e9dd363e89f5b7cfcc228eb0287132.jpg
/instagram_7179986a23f921d3c511f1b571feb9fb.jpg
/instagram_5100564f1ed38d66008e28fdcd3208e8.jpg
/instagram_e4c339c4df7fb7a9b4c2fcccc028a7ec.jpg
/instagram_8500ad90f962dfcde67a0c02df7a0c60.jpg
/instagram_d5fdd444e51262a2265a6d64df2c0d13.jpg
/instagram_df54bfb01de77c4b00717e139aedaae3.jpg
/instagram_4bc73417478de00a3cc220fabef1cd61.jpg
/instagram_3dcb6eb2effb8ab17c99342b7f0e5596.jpg
/instagram_764394a9ec1cd82ea071cc1c7e3d9379.jpg
/instagram_ff042a7477bb5278b9d8dbe99cc8a227.jpg
/instagram_0c5e127e37e3c8715dd4107042727beb.jpg
/wordpress_bc37d85da7ca28e6ae53f3f8118a634a.jpg
/wordpress_187ba387d8b507ea5002b1753934b4fd.jpg
/wordpress_a1165bcc91477d2ff01e98491bcfb09c.jpg
/wordpress_5dbf36f840e5516c6a29b4dbf3b6981f.jpg
/wordpress_41fa6e6889921818458fa093a5bc1d42.jpg
/wordpress_dc34af465e79ff174df0ab70c66642d8.jpg
/wordpress_db044f04e876f6b2e5cce6cbb027b609.jpg
/wordpress_3899b825c8e4cb861c36304c89974dbf.jpg
/wordpress_28690e3cba98e99db49c13c5313e00f6.jpg
/wordpress_26f0a0028a0b0cf34cfd90ec2c5c4e2c.jpg
```

```
/wordpress_75753fbe1812ccdcdc4e292de17d403d.jpg
/wordpress_c9d61bf37645cd7e8958550e064799ca.jpg
/wordpress_b3a9bac1df6f530b9b236ecf613b1c31.jpg
/wordpress_8c0527e95a382c2c27c1b39420df4ef5.jpg
/wordpress_6d4273e21cf5787c29b32aa59d518294.jpg
/wordpress_7939bff981195418eacdb810301c0076.jpg
/wordpress_341b8e71b11a9ea0bb084c1663361673.jpg
/wordpress_0360aea1df79f6306415c1156f8ddbfa.jpg
/wordpress_b9ce93d71036e67306d25057e5d9421c.jpg
/wordpress_21cf8b353aa1298ab6eca17914fdab49.jpg
/wordpress_7f1914445194daa7c492bb2a08ca9d06.jpg
/wordpress_2a78d636a71b2cb875698e6f15760ea8.jpg
/wordpress_6c647c3ec57a3d8ac7d31647c2ce0210.jpg
/wordpress_7895d3950a5c3be1da5716d48663384a.jpg
/wordpress_9da30ece94f7577d5711e45ca1b1cc34.jpg
/wordpress_0b668a07b9ae275ecc77346fced3dd24.jpg
/wordpress_a702cb6013162c2b2d2d8f4c29e14c2d.jpg
/wordpress_3a844a336171480a8c0c3c128ceb7bad.jpg
/wordpress_b80dc650dcbfa9372875b939e2d96850.jpg
/wordpress_e264546a7dcea50935cec3430a3908f5.jpg
/wordpress_afb459190caf559b50b3d2d9475f2d81.jpg
/wordpress_837dbe6e58dfb784dcc3bdac65e1b1e3.jpg
/wordpress_a2982a71cdeb6f7de1ca376236a1b133.jpg
/wordpress_11a98c010479f2afb7e4272a59a1c05f.jpg
/wordpress_7b0d462e407795d453bee70ca301b2b0.jpg
/wordpress_b2cb27bc3846962a580f3d8e7d3230c5.jpg
/wordpress_d4abbcc63857aa22bd7d612586a1f592.jpg
/wordpress_50d8a3f8d0a6c0630658178f2f9961ec.jpg
/wordpress_1400b5e2c9a57a80c1f46d8f194d9fa8.jpg
/wordpress_01dd42242c0efb2b3e4b3af185e59416.jpg
/wordpress_6065a969314bfe3f7faea29fa6a541a9.jpg
/wordpress_8e60371d09a671dc2a6661a82376fb6d.jpg
/wordpress_ff2a0ffd5a28fe1c749a7b1569c48446.jpg
/wordpress_426605c2648908f33407bc41af57b67b.jpg
/wordpress_82bb969f87e9605ccac2b2f4a23e5767.jpg
/wordpress_447ff6c8a79287647a059bb10c6fe768.jpg
/wordpress_79af4201479b28ccc11b13cbc48f1505.jpg
/wordpress_fa4b3ae3dd3a2fa8fce588b1893c7686.jpg
/wordpress_f55f4dc2f3f4af3126453aa32017bb22.jpg
/wordpress_319dafaaf483155a75efd1524652b213.jpg
```

Appendix G

MD5 hash comparison of extracted and original files

Filename	MD5 hash of file on guest	MD5 hash of extracted file
blogspot_06bffe74611bdd2b9ff872ddbce570ad.png	3a5f0ed621d4a91e71d1552c3f43d839	3a5f0ed621d4a91e71d1552c3f43d839
blogspot_468934d7f5ad6d9c7774a838404dc985.png	9666bf35aefe281680ad1772e8315075	9666bf35aefe281680ad1772e8315075
blogspot_675318a2779027329cd8ac494885c41f.png	5f0b3374207ba60eaeb67d54405cbbb2	5f0b3374207ba60eaeb67d54405cbbb2
blogspot_ff0a81287436e721541243946676c9d6.png	af0d55bd006d25f78af9c93218d900b0	af0d55bd006d25f78af9c93218d900b0
facebook_06b0f3f803bf320ba90b0b010c3068b2.jpg	dabab61e0c2dc556b562bba7cb44aea5	dabab61e0c2dc556b562bba7cb44aea5
facebook_513e5dc74b2157cf435831158eca858e.jpg	2c3ee5eba1e821c64c061a73679c8099	2c3ee5eba1e821c64c061a73679c8099
facebook_51e2663a88befc0126b12ffddb896690.jpg	4ec52c217e28079d102370583a81f1f3	4ec52c217e28079d102370583a81f1f3
facebook_53aa521e7c37e20d26ee118564b0c11c.jpg	128b22f4e7803100c19fe16bdb4a0979	128b22f4e7803100c19fe16bdb4a0979
facebook_656c2abe53056907e813d74a11aea29f.jpg	c6eb27ccec0ef100f79613ecdd2b69a5	c6eb27ccec0ef100f79613ecdd2b69a5
facebook_82e71903da34a2f7b95af567e3041333.jpg	118c202de0a31398952703f3b8d5f779	118c202de0a31398952703f3b8d5f779
facebook_ccf11ad50a73c34da3a91ac7a03f82b9.jpg	6338ad48780d7ef1e6fba4a2164a37a2	6338ad48780d7ef1e6fba4a2164a37a2
instagram_0c5e127e37e3c8715dd4107042727beb.jpg	e83c3fb60c665158823c677d68e156f7	e83c3fb60c665158823c677d68e156f7
instagram_107a12d31c387d2d9ea74a997330b261.jpg	231f044800b4f02de119fdd92062ed82	231f044800b4f02de119fdd92062ed82
instagram_2bcfb46076d1cfcaefc2ceaaa0008fae.jpg	02fc3193c900ece13d5f0bcb8033622e	02fc3193c900ece13d5f0bcb8033622e
instagram_3c57861b831cf75b434b577dc7a36741.jpg	288fe0f1219ebe1ae67a1d6a78079132	288fe0f1219ebe1ae67a1d6a78079132
instagram_764394a9ec1cd82ea071cc1c7e3d9379.jpg	1c7f7444f5a3c0c7c21db5d95d99e685	1c7f7444f5a3c0c7c21db5d95d99e685
instagram_779f652d652005665da15f37bb761276.jpg	6644c768c3519d6b64e9d9b44fcf0438	6644c768c3519d6b64e9d9b44fcf0438
instagram_94737932872d5f72d0ac53bf41b867f5.jpg	9a9033b50b1e1092dc751cea217183b8	9a9033b50b1e1092dc751cea217183b8
instagram_a4c033132eb120f377d44a6f5c8879a2.jpg	50aa599173f35fab7488d5a917f740b9	50aa599173f35fab7488d5a917f740b9
instagram_a6c5403844eb25ce118ea0367d66ba81.jpg	42981781f3701c87c716d9c063222e6d	42981781f3701c87c716d9c063222e6d
instagram_b672b724b3417d93bfcbd6d5bf425733.jpg	34684bd96f2de1e03a1cd69d1703f078	34684bd96f2de1e03a1cd69d1703f078
instagram_bb94b0324877d2daf7ec893075cb0824.jpg	c85bd222c7d71cf092943c0064240d7e	c85bd222c7d71cf092943c0064240d7e
instagram_bd5260c95858fbd196baf2e68575ab77.jpg	ce31a62f5859f958315a57de5f2834ff	ce31a62f5859f958315a57de5f2834ff
wordpress_41fa6e6889921818458fa093a5bc1d42.jpg	d01f8a460f11316a86f537cd01a0a972	d01f8a460f11316a86f537cd01a0a972
wordpress_6d4273e21cf5787c29b32aa59d518294.jpg	cf153db9aa806c9f95246d78c174572a	cf153db9aa806c9f95246d78c174572a
wordpress_75753fbe1812ccdcdc4e292de17d403d.jpg	ab7d32fc8769672028856211173de616	ab7d32fc8769672028856211173de616
wordpress_7f1914445194daa7c492bb2a08ca9d06.jpg	654939d8946815ef1853ac22b09496d6	654939d8946815ef1853ac22b09496d6
wordpress_8c0527e95a382c2c27c1b39420df4ef5.jpg	56ea25048ef3b527ec2631568ebe9b44	56ea25048ef3b527ec2631568ebe9b44
wordpress_afb459190caf559b50b3d2d9475f2d81.jpg	4b11a0e82806bc7b3382d2cb10558d36	4b11a0e82806bc7b3382d2cb10558d36
wordpress_c9d61bf37645cd7e8958550e064799ca.jpg	b2bad70baed422da8a6fe569ac5efb8e	b2bad70baed422da8a6fe569ac5efb8e

Appendix H

SQL output showing collected information

```
sqlite> select external_ip,local_port,filepath from
conntrack,filetrack,files where conntrack.id = filetrack.cid AND
filetrack.fid = files.id AND filepath NOT NULL;
192.168.122.1|443|/root/dump/b_675318a2779027329cd8ac494885c41f.png
192.168.122.1|443|/root/dump/b_468934d7f5ad6d9c7774a838404dc985.png
192.168.122.1|443|/root/dump/b_06bffe74611bdd2b9ff872ddbce570ad.png
192.168.122.1|443|/root/dump/b_ff0a81287436e721541243946676c9d6.png
192.168.122.1|443|/root/dump/f_513e5dc74b2157cf435831158eca858e.jpg
192.168.122.1|443|/root/dump/f_82e71903da34a2f7b95af567e3041333.jpg
192.168.122.1|443|/root/dump/f_ccf11ad50a73c34da3a91ac7a03f82b9.jpg
192.168.122.1|443|/root/dump/f_51e2663a88befc0126b12ffddb896690.jpg
192.168.122.1|443|/root/dump/f_06b0f3f803bf320ba90b0b010c3068b2.jpg
192.168.122.1|443|/root/dump/f_53aa521e7c37e20d26ee118564b0c11c.jpg
192.168.122.1|443|/root/dump/f_656c2abe53056907e813d74a11aea29f.jpg
192.168.122.1|443|/root/dump/i_bb94b0324877d2daf7ec893075cb0824.jpg
192.168.122.1|443|/root/dump/i_bd5260c95858fbd196baf2e68575ab77.jpg
192.168.122.1|443|/root/dump/i_2bcfb46076d1cfcaefc2ceaaa0008fae.jpg
192.168.122.1|443|/root/dump/i_a6c5403844eb25ce118ea0367d66ba81.jpg
192.168.122.1|443|/root/dump/i_b672b724b3417d93bfcbd6d5bf425733.jpg
192.168.122.1|443|/root/dump/i_779f652d652005665da15f37bb761276.jpg
192.168.122.1|443|/root/dump/i_94737932872d5f72d0ac53bf41b867f5.jpg
192.168.122.1|443|/root/dump/i_3c57861b831cf75b434b577dc7a36741.jpg
192.168.122.1|443|/root/dump/i_a4c033132eb120f377d44a6f5c8879a2.jpg
192.168.122.1|443|/root/dump/i_107a12d31c387d2d9ea74a997330b261.jpg
192.168.122.1|443|/root/dump/i_764394a9ec1cd82ea071cc1c7e3d9379.jpg
192.168.122.1|443|/root/dump/i_0c5e127e37e3c8715dd4107042727beb.jpg
192.168.122.1|443|/root/dump/w_41fa6e6889921818458fa093a5bc1d42.jpg
192.168.122.1|443|/root/dump/w_75753fbe1812ccdcdc4e292de17d403d.jpg
192.168.122.1|443|/root/dump/w_c9d61bf37645cd7e8958550e064799ca.jpg
192.168.122.1|443|/root/dump/w_8c0527e95a382c2c27c1b39420df4ef5.jpg
192.168.122.1|443|/root/dump/w_6d4273e21cf5787c29b32aa59d518294.jpg
192.168.122.1|443|/root/dump/w_7f1914445194daa7c492bb2a08ca9d06.jpg
192.168.122.1|443|/root/dump/w_afb459190caf559b50b3d2d9475f2d81.jpg
```

Appendix I

Dir list of extracted files

```
ls -1 /root/dump/
b_675318a2779027329cd8ac494885c41f.png
b_06bffe74611bdd2b9ff872ddbce570ad.png
b_468934d7f5ad6d9c7774a838404dc985.png
b_ff0a81287436e721541243946676c9d6.png
f_513e5dc74b2157cf435831158eca858e.jpg
f_82e71903da34a2f7b95af567e3041333.jpg
f_ccf11ad50a73c34da3a91ac7a03f82b9.jpg
f_51e2663a88befc0126b12ffddb896690.jpg
f_06b0f3f803bf320ba90b0b010c3068b2.jpg
f_53aa521e7c37e20d26ee118564b0c11c.jpg
f_656c2abe53056907e813d74a11aea29f.jpg
i_bb94b0324877d2daf7ec893075cb0824.jpg
i_bd5260c95858fbd196baf2e68575ab77.jpg
i_2bcfb46076d1cfcaefc2ceaaa0008fae.jpg
i_779f652d652005665da15f37bb761276.jpg
i_a6c5403844eb25ce118ea0367d66ba81.jpg
i_b672b724b3417d93bfcbd6d5bf425733.jpg
i_107a12d31c387d2d9ea74a997330b261.jpg
i_3c57861b831cf75b434b577dc7a36741.jpg
i_94737932872d5f72d0ac53bf41b867f5.jpg
i_a4c033132eb120f377d44a6f5c8879a2.jpg
i_0c5e127e37e3c8715dd4107042727beb.jpg
i_764394a9ec1cd82ea071cc1c7e3d9379.jpg
w_41fa6e6889921818458fa093a5bc1d42.jpg
w_75753fbe1812ccdcdc4e292de17d403d.jpg
w_6d4273e21cf5787c29b32aa59d518294.jpg
w_7f1914445194daa7c492bb2a08ca9d06.jpg
w_8c0527e95a382c2c27c1b39420df4ef5.jpg
w_afb459190caf559b50b3d2d9475f2d81.jpg
w_c9d61bf37645cd7e8958550e064799ca.jpg
```

Bibliography

[1] Torbjörn Petterson. Cryptographic key recovery from Linux memory dumps, 2007.

[2] Freddie Witherden. Memory Forensics over the IEEE 1394 Interface, 2010.

[3] Huiming Yu, Nakia Powell, Dexter Stembridge and Xiaohong Yuan. Cloud computing and security challenges, ACM-SE 12 Proceedings of the 50th Annual Southeast Regional Conference, 2012

[4] Tom McCafferty. XenServer 6.1 - Growing Market Share and Leading the Second Source Hypervisor Trend, http://blogs.citrix.com/2012/10/03/xenserver-6-1-growing-market-share-and-leading-the-second-source-hypervisor-trend, 2012, accessed February 2014

[5] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing, National Institute of Standards and Technology, 2011

[6] Bob Matthews and Norm Murray. Virtual Memory Behavior in Red Hat Linux Advanced Server 2.1, Red Hat

[7] http://www.fbi.gov/about-us/history/brief-history, accessed February 2014

[8] James Poore, Juan Carlos Flores and Travis Atkison. Evolution of digital forensics in virtualization by using virtual machine introspection, Proceedings of the 51st ACM Southeast Conference Article No. 30, 2013

[9] Volatility introduction, https://code.google.com/p/volatility/wiki/VolatilityIntroduction, accessed February 2014

[10] Tal Garfinkel and Mendel Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection, Proceedings of the Network and Distributed Systems Security Symposium, 2003

[11] Bryan D. Payne, Martim D. P. de A. Carbone and Wenke Lee, Secure and Flexible Monitoring of Virtual Machines, Georgia Institute of Technology, 2007

[12] LibVMI introdution, https://code.google.com/p/vmitools/wiki/LibVMIIntroduction, accessed February 2014

[13] http://www.alexa.com/topsites, accessed March 2014

[14] Scott Radvan and Tahlia Richardson. Red Hat Enterprise Linux 7.0 Beta Virtualization Security Guide, Red Hat, 2014

[15] Karen Scarfone, Murugiah Souppaya and Paul Hoffman. Guide to Security for Full Virtualization Technologies, National Institute of Standards and Technology, 2011

[16] Virtualization Special Interest Group. PCI DSS Virtualization Guidelines, PCI Security Standards Council, 2011

[17] Securing Debian Manual, http://www.debian.org/doc/manuals/securing-debian-howto/index.en.html, 2013, accessed March 2014

[18] http://www.hpl.hp.com/research/linux/httperf, accessed March 2014

[19] Xen, https://help.ubuntu.com/community/Xen, accessed February 2014

[20] libVMI installation, https://code.google.com/p/vmitools/wiki/LibVMIInstallation, accessed February 2014

[21] Volatility installation, https://code.google.com/p/volatility/wiki/VolatilityInstallation, accessed February 2014

[22] HTTPD - Apache2 Web Server, https://help.ubuntu.com/10.04/serverguide/httpd.html, accessed February 2014

[23] Linux Memory Forensics, https://code.google.com/p/volatility/wiki/LinuxMemoryForensics, accessed March 2014